CHOReOS Dynamic Development Model Definition (D2.1)

Marco Autili, Davide Di Ruscio, Inverardi Paola, Tivoli Massimo, Dionysis Athanasopoulos, Apostolos Zarras, Panos Vassiliadis, James Lockerbie, Neil Maiden, Antonia Bertolino, et al.

ICT IP Project

Deliverable D2.1

CHOReOS Dynamic Development Model Definition

http://www.choreos.eu

LaTeX template v. 1.14

Project Number	:	FP7-257178
Project Title	:	CHOReOS Large Scale Choreographies for the Future Internet

Deliverable Number	:	D2.1
Title of Deliverable	:	CHOReOS Dynamic Development Model Definition
Nature of Deliverable	:	Report
Dissemination level	:	Public
Licence	:	Creative Commons Attribution 3.0 License
Version	:	A.0
Contractual Delivery Date	:	30 September 2011
Actual Delivery Date	:	14 October 2011
Contributing WP	:	WP2
Editor(s)	:	Marco Autili (UDA), Massimo Tivoli (UDA)
Author(s)	:	Marco Autili (UDA), Davide Di Ruscio (UDA), Paola Inverardi (UDA), Massimo Tivoli (UDA), Dionysis Athanasopoulos (UOI), Apostolos Zarras (UOI), Panos Vassiliadis (UOI), James Lockerbie (CITY), Neil Maiden (CITY), Antonia Bertolino (CNR-ISTI), Guglielmo De Angelis (CNR-ISTI), Amira Ben Amida (Petals Link), Darius Silingas (No Magic Europe), Rokas Bartkevicius (No Magic Europe), Marco A. Gerosa (USP), Gustavo A. Oliva (USP), Leonardo Leite (USP), Guilherme M. Nogueira (USP), Alfredo Goldman (USP), Yanik Ngoko (USP)
Reviewer(s)	:	Valérie Issarny (INRIA), Hugues Vincent (THALES), Antonia Bertolino (CNR)

Abstract

The Future Internet envisions a ubiquitous world where available services can be easily discovered and coordinated so as to fit users needs. Service choreographies will play a central role in this vision as an effective means to allow heterogeneous services to suitably collaborate. This deliverable defines the *CHOReOS Dynamic Development Process Model* by refining and completing the artefacts/activities and the relationships among them already sketched into the DoW (under the WP2 description), and conceptualized by the the CHOReOS *conceptual model* [CHO11b]. The process *model* is an abstract and simplified description of what will be the *actual* CHOReOS software development process to be defined at M24. The process model describes the "strategy" to be used during the choreography life cycle from design, to development, to maintenance (and hence from static, to runtime, to evolution). The model is made up of activities, common to (almost) every process, but structured in a particular way (i.e., the "CHOReOS way"), hence distinguishing the CHOReOS development process from others.

Keyword List

Dynamic Software Development Process Model, Conceptual Model, Requirements Specification, Large-scale Service Base Management, Meta-modeling, Choreography-centric Service Oriented Computing and Architecture, Model-driven Engineering, BPMN2, Dependency Oriented Choreography Analysis

Document History

Version	Changes	Author(s)
0.1	Outline Draft	Marco Autili, Davide Di Ruscio, Paola Inverardi, Massimo Tivoli (UDA)
1.0	Final Outline	Marco Autili, Davide Di Ruscio, Massimo Tivoli (UDA)
1.1	First draft of Introduction, Case-Study, Development Process Model, and Choreography Synthesis	Marco Autili, Massimo Tivoli (UDA)
1.2	Refined draft of Introduction, Case-Study, Development Process Model, and Choreography Synthesis to allow other partners to provide already integrated contributions	Davide Di Ruscio, Massimo Tivoli (UDA)
1.3	Added Section “Deployment and run-time ULS choreographies” into Chapter 2, and Chapter 5.	Marco Autili, Davide Di Ruscio (UDA)
1.4	Added contributions to Chapter 3.	Marco Autili, Davide Di Ruscio, Massimo Tivoli (UDA), James Lockerbie, Neil Maiden (CITY)
1.5	Added contributions to Chapter 4 and Chapter 5.	Marco Autili, Davide Di Ruscio, Massimo Tivoli (UDA), Dionysis Athanasopoulos, Apostolos Zarras, Panos Vassiliadis (UOI), Antonia Bertolino, Guglielmo De Angelis (CNR-ISTI), Amira Ben Amida (Petals Link), Marco A. Gerosa, Gustavo A. Oliva, Leonardo Leite, Alfredo Goldman, Yanik Ngoko (USP)
2.0	Integration of final contributions by all have been	Marco Autili, Davide Di Ruscio, Massimo Tivoli, Paola Inverardi (UDA), Darius Silingas, Rokas Bartkevicius (No Magic Europe)
2.1	Introduction, Table of Content, and Chapter 2 have been revisited according to the comments of the internal reviewers.	Marco Autili, Davide Di Ruscio, Massimo Tivoli (UDA), Zarras Apostolos, Panos Vassiliadis, Dionysis Athanasopoulos (UOI), James Lockerbie, Neil Maiden (CITY), Antonia Bertolino, Guglielmo De Angelis (CNR-ISTI)

2.2	The whole deliverable has been revisited according to the comments of the internal reviewers.	Marco Autili, Davide Di Ruscio, Massimo Tivoli (UDA)
3.0	Final version.	Marco Autili, Davide Di Ruscio, Massimo Tivoli (UDA)

Document Reviews

Review	Date	Ver.	Reviewers	Comments
Outline	16 May 2011	1.0	All authors	Outline agreed by all
Draft	01 September 2011	2.0	Marco Autili, Paola Inverardi, Davide Di Ruscio, Massimo Tivoli	Complete version released for review
QA	10 October 2011	3.0	Marco Autili, Paola Inverardi, Valérie Issarny, Hugues Vincent, Antonia Bertolino	Internal review comments
PTC	14 October 2011	A	PTC	Internal review comments addressed and final version ready

Glossary, acronyms & abbreviations

Item	Description
BPMN2	Business Process Modelling Notation, Version 2 - OMG
CA	Consortium Agreement
CTT	Concurrent Task Trees
DL	Deliverable Leader
DOW	Description of Work
FI	Future Internet
IAC	Industrial Advisory Committee
M2C	Model-to-Code
M2M	Model-to-Model
MDD	Model Driven Development
MDE	Model Driven Engineering
MST	Management Support Team
OSS	Open Source Software
PL	Project Leader
PMT	Project Management Committee
PO	Project Officer
PTC	Project Technical Committee
QoS	Quality of Service
SL	Scientific Leader
SNA	Social Network Analysis
ULS	Ultra Large Scale
WP	Work Package
WPL	Work Package Leader

Table Of Contents

List Of Tables

List Of Figures

1 Introduction

The near future in software production envisions a ubiquitous world of available services that can be easily discovered and coordinated to fit users' needs. The Internet of Services paradigm [COR] emerges from the convergence of the Future Internet (FI) and of the Service-Oriented Software paradigm. Services play a central role in this vision as effective means to achieve interoperability between heterogeneous parties of a business process and independence from the underlying infrastructure. At the same time they offer an open platform to build new value added service-based systems as a *choreography* of available services discovered within the FI.

The Ultra-Large-Scale (ULS), pervasiveness, decentralization, and heterogeneity of the FI call for adaptable, QoS-aware highly-scalable choreographies. This demands new service collaboration and coordination paradigms that span the entire choreography life-cycle from requirements specification to deployment, execution, and evolution.

In particular, choreographies must meet users' requirements and be adaptable to context changes (e.g., with respect to user- and resource-centric data, and QoS). To this end, further accounting for the FI challenges, certain phases of the development process together with their proper artefacts, need to become pervasive, accompanying the choreography implementation at run-time [BBF09].

This deliverable presents a first version of the CHOReOS dynamic development process model. One of the main objectives of CHOReOS is to define a domain-expert-centric process for the development of ULS choreographies in FI. Towards the definition of this dynamic development process, WP2 identifies the following (macro-)activities. Due to the different amount of resources required for their development, as pointed out below, some of them are developed in other WPs:

1) A domain-expert requirements specification of adaptable QoS-aware highly-scalable choreographies (developed in WP2);
2) Large scale abstraction-oriented service base management (developed in WP2);
3) Automated choreography synthesis (developed in WP2);
4) Choreography deployment and execution (developed in WP3);
5) Design and run-time analysis (developed in WP2);
6) Governance V&V, monitoring and V&V configuration (developed in WP4).

Note that a comprehensive discussion of the State-Of-The-Art for all the above listed activities has been carried on in Deliverable D1.1 [Tea10] and, hence, the interested reader can refer to it.

Strictly concerning the purpose of this deliverable, in this document we propose a model that abstractly characterizes the CHOReOS software development process by identifying the main activities (their flow and manipulated artefacts) that need to be performed to develop *CHOReOS choreographies*. Since activities (3) and (5) rely on the outcomes of activities (1) and (2), their development will start at M12. Thus, according to the project schedule and WPs structure, in this deliverable, more attention is devoted to the first two activities.

This deliverable is organized as follows. We start with a description of the development process model in Chapter 2, where all the above mentioned activities and related artefacts are integrated into an overall picture. In particular, capitalizing on the CHOReOS conceptual model described in Deliverable D1.2 [CHO11b], the refined view of the development process model provides a detailed description of the flow of process activities and manipulated data, and of the specific methodologies adopted by each activity (Sections 2.1 to 2.5). Then, Chapters 3 and 4 provide a more detailed description of the first two activities by considering the running passenger friendly airport case study (based on the WP6 use case). Chapter 5 discusses analysis mechanisms for QoS-aware ULS choreographies, which include a QoS prediction model to support scalability and a dependency-centric analysis to support change impact analysis. Conclusions and future work are given in Chapter 6.

2 CHOReOS Dynamic Development Process Model

This chapter extends and revises the model of the development process we have preliminarily defined in the DoW, and defines it as BPMN2 Process Diagrams. A model of a development process is an abstract and simplified description of what will be the actual process. The CHOReOS development process model describes the "strategy" that CHOReOS uses for specifying, analyzing, enacting, and monitoring ULS choreographies during the whole life cycle (from static to runtime to evolution). The model is made up of activities, common to (almost) every development process, but structured in a particular way (i.e., the "CHOReOS way"), hence distinguishing the CHOReOS development process from others.

We give a high level description that characterizes the CHOReOS software development process so that the reader grasps what are the main activities that need to be performed and what are the artefacts manipulated by these activities without referring to specific technologies, tools, standards, models, etc.

Broadly speaking, a "standard" development process focuses on activities that are traditionally divided into design-, deployment-, and run-time activities. The evolutionary nature of choreographies in the FI makes unfeasible a standard development process since, e.g., dealing with adaptation and QoS-awareness would require predicting functional and non-functional properties of the choreography (with respect to virtually any possible change) before its execution, and representing these properties in terms of suitable artefacts (e.g., models) that can be exploited at run-time (e.g., for monitoring purposes). That is, whenever a change occurs, if choreography evolution has to be supported by means of adaptation, all the artefacts and models might be exploited also by the deployment- and run-time activities hence leading to a "non-standard" development process view, i.e., a dynamic development process model. Central to the development process is the notion of models. Models are abstract views of systems, suitable for reasoning, developing, and validating a real system. Models can be functional and non-functional and can represent different levels of abstractions of the real system, from requirements to code. Model Driven Development (MDD) and, in particular, Model-to-Model (M2M) transformations enable the shift among various levels of abstractions and across several domains, possibly in an automatic way. The models@runtime [BBF09] approach seeks to extend the applicability of models produced in MDD approaches to the run-time environment, hence promoting even more the realization of the dynamic development process view.

Figure 2.1 shows a revised version of the CHOReOS Development Process Model, preliminarily sketched into the DoW under the WP2 description. In this model we do not specify who are the involved stakeholders and who does which activity, since such details are given in the deliverables D5.1 [Tea11a] and D5.2 [Tea11b], which specifies the CHOReOS IDRE supporting the CHOReOS development process.

- First, as depicted in Figure 2.1, CHOReOS intends to support the systematic development of choreographies from their design to their actual enactment and execution. CHOReOS in particular investigates techniques and tools that emerge from Model Driven Engineering (MDE) and generative programming research areas [CE00], and makes use of two orthogonal transformational approaches: (i) a *top-down* transformation process and (ii) a *cross-cutting* transformation process. The former will serve to refine the structured natural language and *i**-based domain-expert requirements specification (see Chapter 3) into BPMN2-based choreography model(s) (BPMN2 Choreography Diagrams and/or BPMN2 Collaboration Diagrams). This model can be in turn fur-

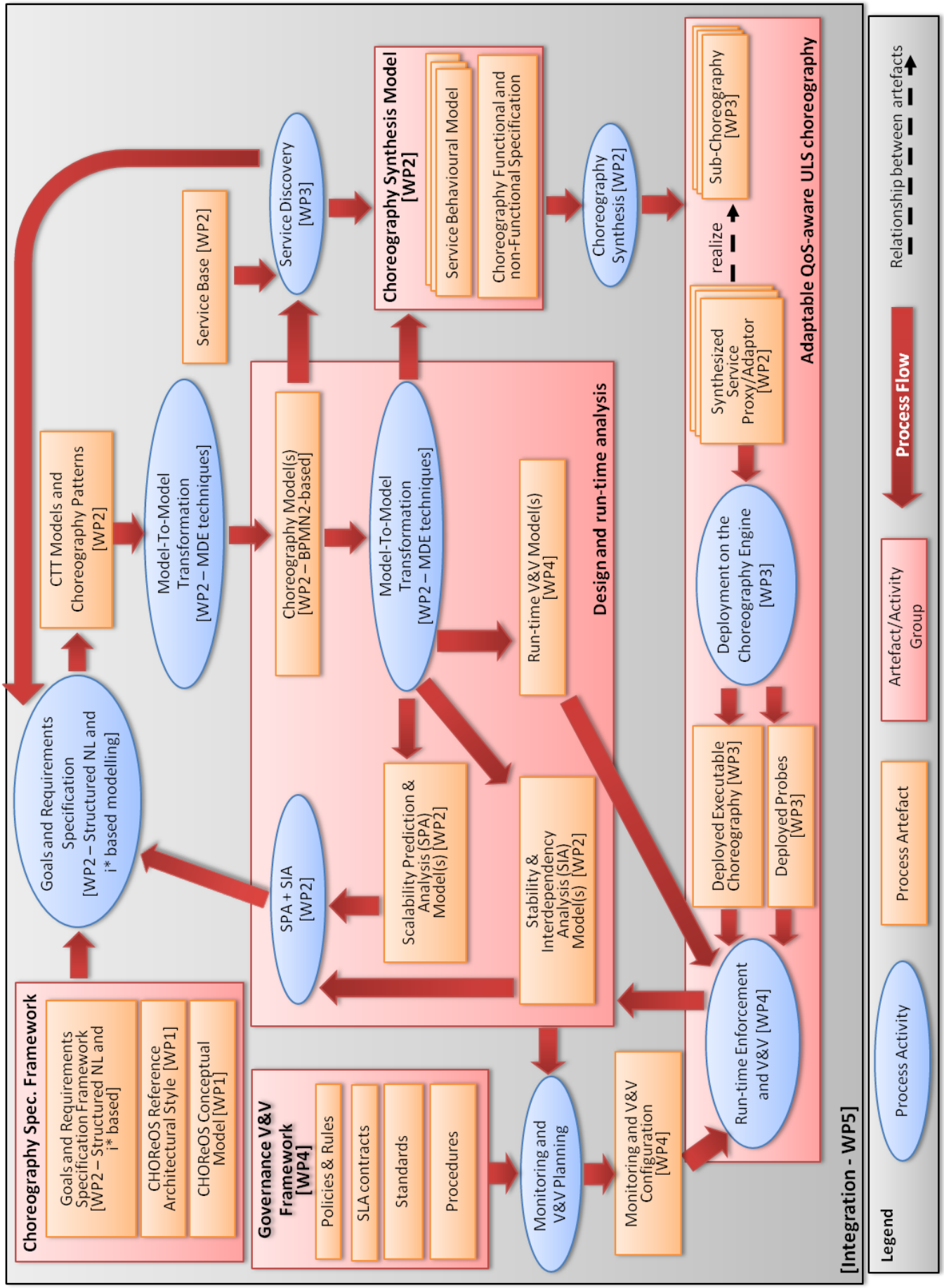


Figure 2.1: CHOReOS Development Process Model

ther transformed into technique-specific models (e.g., models suitable for analysis and synthesis purposes). The latter serves to integrate the different modeling/reasoning technologies by passing from a technique-specific model to a different technique-specific model. This will bridge the gap between the various models that have to be used for choreography synthesis, analysis, validation, and implementation purposes.

- A distributed (large scale) service base further manages information about available services offered by service providers. As detailed in Chapter 4, abstractions-oriented service base organizes available services into functional and non-functional views. Each view is characterized by corresponding abstractions (functional/non-functional) and by a set of available services that are represented by each abstraction.

- The approach envisioned for the choreography synthesis then starts from (i) the BPMN2-based choreography model(s) and from (ii) the set of services discovered from the large scale service base. As detailed in Chapter 3, input (i) comes from the refinement of the domain-expert requirements specification obtained by means of the top-down transformational process. Input (ii) comes from the exploitation of the service base management mechanisms. As detailed later, the synthesis process assumes that the services in the registry/base have been discovered so that they satisfy the local (to the service) functional and non-functional requirements that have been specified for the choreography and, hence, can be considered as potential candidates to participate in the global choreography process. The choreography synthesis produces (possibly with partial human intervention/inspection) software entities that, on top of the CHOReOS service-oriented middleware, support the execution of choreographies in a distributed way.

- As detailed in Chapter 5, design and run-time analysis activities are performed to assess if the choreography can evolve and scale, while keeping some performance parameters, such as QoS attributes. To achieve these goals, the CHOReOS dynamic process must continuously monitor and predict choreography QoS, compute choreography stability, and measure the dependency among its services. The stability and interdependence analysis may also be performed at design-time, helping the creation of a choreography design with low coupling among its services.

Leveraging the models@runtime approach [BBF09], the CHOReOS dynamic development process poses new challenges whose solution represents a significant "process-level" progress with respect to the State of the Art [Tea10], also concerning the transformational approaches that we use to integrate the various artefacts and activities, and the progress that we make with respect to the MDD and transformation techniques in the literature. In fact, differently from what is traditionally done, it should be possible for end-users and domain experts, even when they do not have a technical background, to specify the desired choreography with respect to their domain (business) requirements and goals. For this reason, within the CHOReOS development process, a novel key artefact for modelling is represented by the requirements specification framework based on structured natural language and a suitable extension of the *i** notation (see Section 2.1 and Chapter 3). The output in the form of choreography CTT (Concurrent Task Trees) models and choreography patterns, shown in Figure 2.1, represents a very high-level choreography specification which describes temporal relationships among tasks associated with reusable choreography strategies (known choreography-based solutions expressed as patterns). Although, on the one side, this specification allows domain experts to effectively specify the choreography and end-users its requirements, on the other side, automated reasoning cannot be easily performed directly from it. To deal with this issue, M2M transformation techniques are developed to translate this high-level specification into the more technical form of BPMN2-based choreography specification. Moreover, model transformations is developed also to support service discovery (see Section 2.2 and Chapter 4), to derive the choreography models suitable for synthesis purposes (see Section 2.3), for choreography analysis, for choreography monitoring and validation (see Section 2.5), scalability (see Section 5.1), and evolvability (see Section 5.2). Concerning the synthesis purposes, model-to-model transformations will be developed to automatically derive the LTS-based specification

of the choreography from a BPMN2 specification of it. To this end, it is worth to note that BPMN2 can be used for creating simple and rather abstract choreography specifications (that can be useful for providing, e.g., a business manager with a high-level view of a given business process), but also detailed and technical specifications (that can be parsed by a machine and automatically manipulated by developers and analysts, e.g., for tool-supported analysis and synthesis).

As previously said, in the CHOReOS development process different interdependent models are produced, and proper techniques and tools have to be devised in order to deal with synchronization and change propagation issues. In particular, changes occurring in a model can have a strong impact on all the other interoperating models (each of them possibly conforming to different notations). In order to keep models in a consistent state, changes need to be propagated from the updated model to all the others that have to be used for, e.g., choreography analysis, validation, synthesis, and implementation purposes. When dealing with multiple notations, propagating changes may be a complex task; such a task is inevitable and requires to be managed by model (non-necessarily automated) synchronization techniques.

In the following, we will use BPMN2 Process Diagrams as standard notation also for specifying the flow(s) among the CHOReOS development process activities and manipulated data. To this end, Figure 2.2 shows the BPMN2 Process Diagrams concerning *Goals and Requirements Specification*, *Choreography Synthesis*, *Discovery of Service*, *Enactment of Service Choreography*, *Design* and *Run-time Analysis*, *Monitoring* and *V&V Planning*. Such activities will be refined and described in more details later in this chapter. We recall that this process represents a first version that will be refined in parallel with the work undertaken by the others technical WPs, hence leading to its final version to be delivered at M24.

According to Figure 2.2, the *Model-to-Model Transformation* takes as input *CTT Models and Choreography Patterns* as derived by the specification of the choreography goals and requirements (see Section 3). As shown by the feedback-loop between *Discovery of Service* and *Goals and Requirements Specification*, the latter encompasses an initial service discovery based on clustered requirements to find some initial service classes to feed into the CTT models and choreography patterns. The model-to-model transformation process produces a *Choreography Model*, which can be further evaluated by means of *Dependency Analysis* that is part of *Design-time Analysis*.

Such a model is also taken as input by a service choreography synthesizer together with a set of service abstractions (see *Abstractions&Represented Services* in Figure 2.2). The latter model possible concrete services that can be potentially used, at run-time, to realize the choreography. Service abstractions are discovered from the *Abstraction Base* (see Section 4) with respect to the specification given by the choreography model. As further detailed in the following, the output of the service choreography synthesis is a set of *Coordination delegates* that are generated to distributively support choreography enactment, and hence for enabling the choreography realization. Service dependencies, established according to the produced coordination delegates, are then evaluated at run-time in terms of coupling degree and stability (see Chapter 5) in order to improve choreography evolution. Dependable evolution is achieved via run-time monitoring techniques that aim at detecting possible changes in the choreography execution context. When a context change occurs, and it is such that the choreography needs to be redesigned, the entire process is re-iterated hence breaking the usual division among design- and run-time activities. That is, all process activities are performed at run-time. The smooth cooperation among services is controlled and validated according to established policies and rules.

By following the process shown in Figure 2.2, this chapter is organized as follows. Section 2.1 introduces the choreography goals and requirements specification activity, which is further detailed in Chapter 3. Section 2.2 outlines the proposed abstractions-oriented service base management supporting the service discovery concern in the context of CHOReOS. Such activity is then detailed in Chapter 4. Section 2.3 describes the choreography synthesis process. Section 2.4 describes the required run-time support to deploy and enact choreographies. Section 2.5 discusses how to test and monitor choreographies. Section 2.6 introduces the running passenger friendly airport case study that is used to illustrate the description carried on in Chapters 3 and 4. We recall that design and run-time

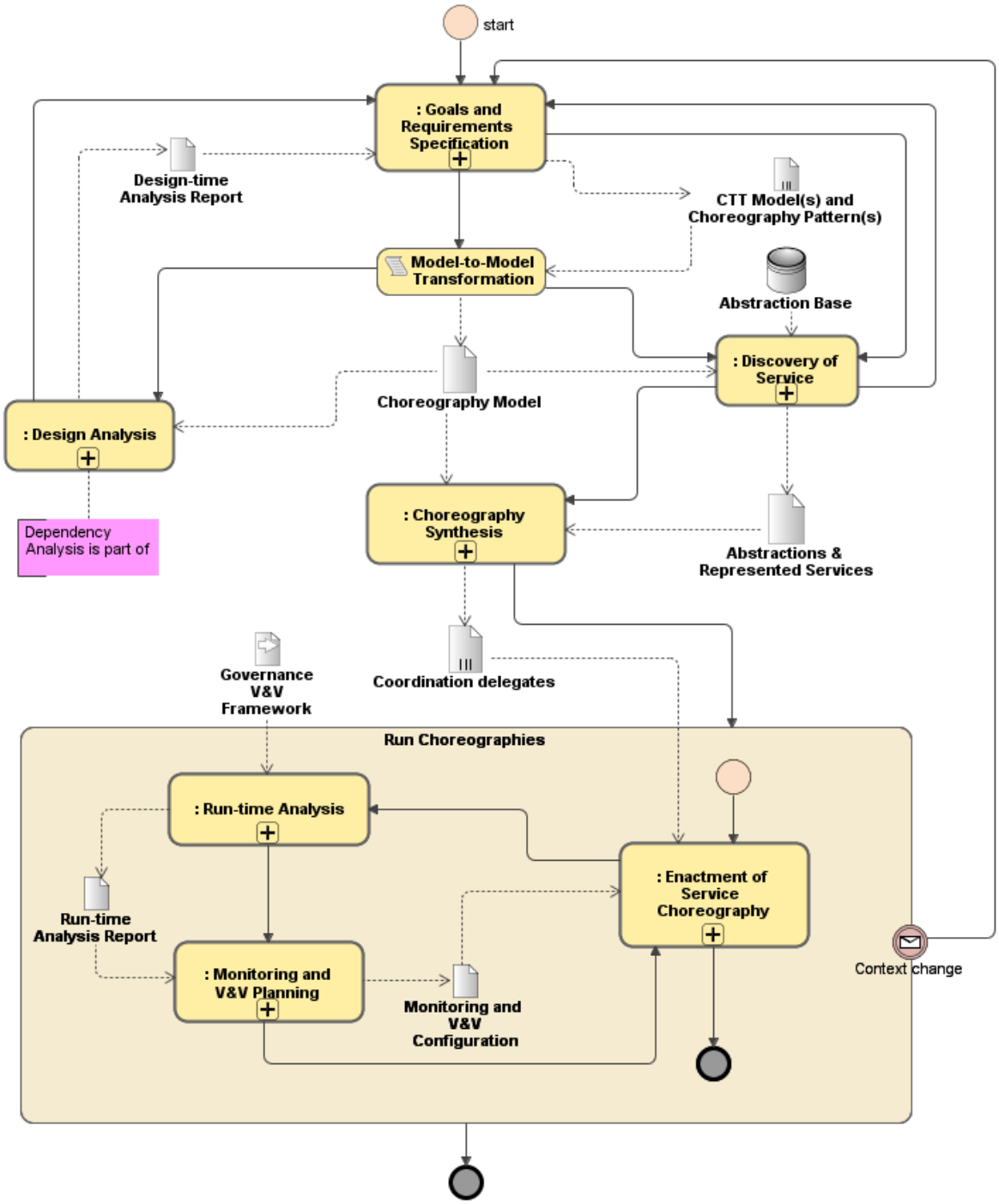


Figure 2.2: From specification to execution

analysis is discussed in Chapter 5.

2.1. Domain Expert Specification of QoS-aware Choreographies

One of the aims of CHOReOS is to enable a domain-expert-centric approach to specifying QoS-aware ULS choreographies, as well as taking an adaptive approach based on the needs of end-users. The approach enables domain experts, rather than requirements analysts, to express functional and quality requirements on services and service-based applications. The domain expert approach also takes input

from service consumers, who express their service needs, in order for the domain expert to reflect the requirements of end users in the specification.

The specification process (see Figure 2.3) begins with the service consumer who provides the primary source of requirements by expressing their service needs. The domain expert, acting in a moderator role, then receives consumer requirements through the evolving requirements specification and can also act as a surrogate for individual service consumers and express their own domain specific requirements. The domain expert can constantly review the latest requirements/queries using tool support to extract the overall requirements on the services to be choreographed. In addition, emerging requirements can be pushed to the domain expert in order to elicit feedback. Different patterns of service needs will emerge based on functionality, quality, and user type or preference - all of which will act as constraints on the services to be choreographed.

2.1.1. Domain expert and service consumer expression of requirements

The role of the service consumer cannot assume domain knowledge or the ability to express needs effectively in a requirements specification. Whilst the domain expert will have domain knowledge, expressing this in the form of requirements using even semi-formal notations supported by the UML may be beyond most of them. Therefore, a core challenge for CHOReOS is to address how requirements on future services can be expressed as simply as possible.

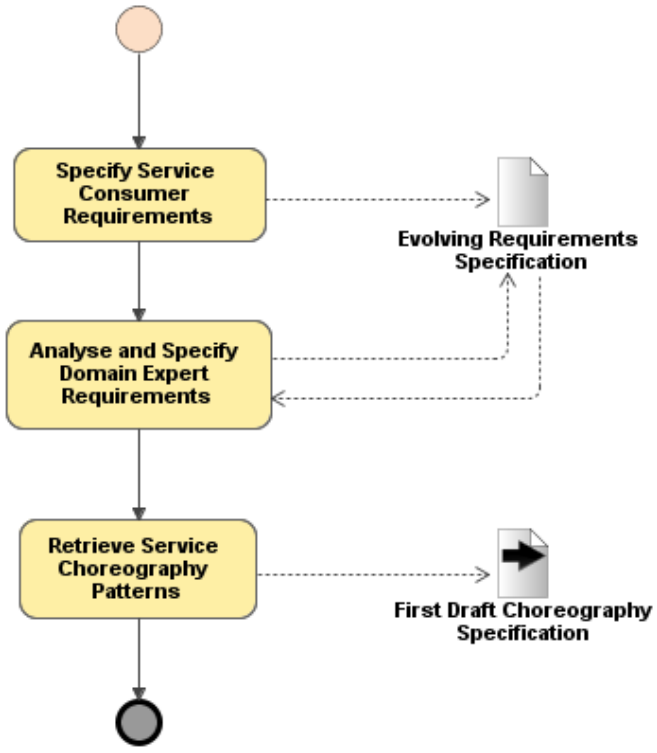


Figure 2.3: Goal and requirements specification

It is expected that service consumers and domain experts will be able to express functional type requirements in natural language. However, the bigger challenge is how to capture quality requirements, as even trained analysts find it difficult to write measurable requirements to express performance, reliability and availability using standard metrics. Therefore, some degree of constraint or structure of free-text expression is needed to capture the service quality needs of the users effectively. To do this we use qualifiers [AS02], which transform the functional root of a requirement into a non-functional requirement (NFR). Given that users might only express functional requirements when requested to describe their service needs, this approach encourages them to express a quality qualifier, or ideally more

than one qualifier per requirement description. Rather than ask the user to express requirements using typical NFR types, our approach prompts them to express qualities on the service through 5 simple questions: ”How quick?”, ”How secure?”, ”How precise?”, ”How robust?”, and ”How simple to use?”.

These questions were selected based on underlying research behind the Service Measurement Index (SMI) model of cloud service characteristics developed for the Cloud Commons Consortium [ZLHM11]. The SMI model specifies definitions, measures, metrics and indicators of cloud service characteristics, along with the structures and relationships that exist between them. For the CHOReOS quality model, research into web service quality ontologies, requirements taxonomies and published system quality models was reapplied to formulate a revised set of qualities and associations suitable for specifying requirements on QoS-aware choreographies. For example, cloud characteristics more specific to consumer organisations such as business agility and risk were removed, and more emphasis was placed on qualities concerning the CHOReOS FI challenges (listed in D1.3 [CHO11d])

Figure 2.4 presents a simple representation of the model, and shows how the qualifier questions map onto service-based application qualities and qualities-of-service on services aggregated in these applications. It would not be realistic for domain experts and service consumers to express requirements on qualities such as scalability, flexibility and serviceability as these areas are the concern of service developers and providers. However, through associations in the model these can be implied, for example, high performance may require a particular level scalability and flexibility of the services.

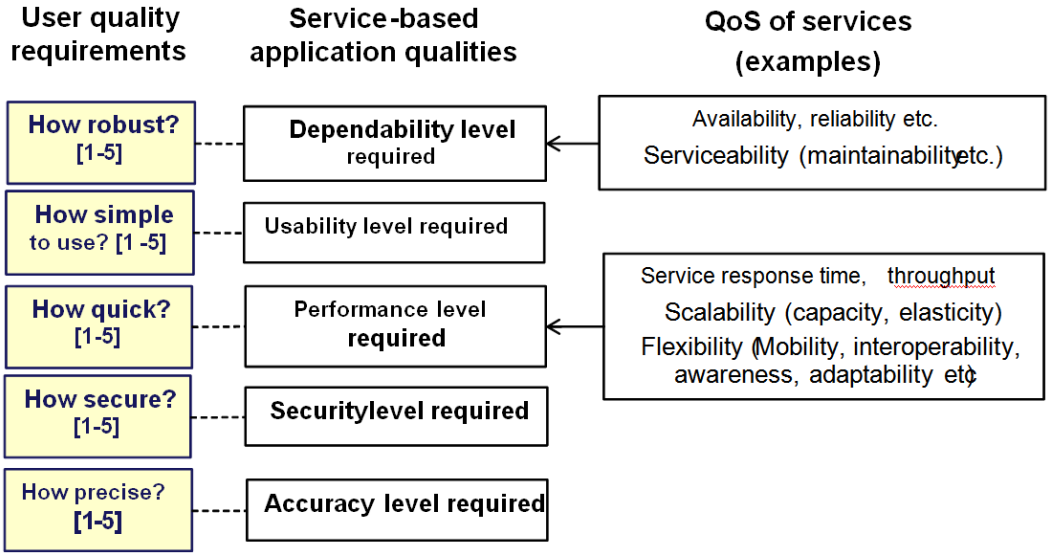


Figure 2.4: Simplified overview of the CHOReOS quality model

As can be seen from Figure 2.4, the user response to the prompts is given on a scale of 1 to 5 in order to prioritise requirements based on the relative importance of the quality qualifiers. Constraints are applied to the user expressed values, for example, only one value of 5 is permissible per requirement. Measures and metrics are hidden behind the user expressed qualifiers, as we cannot assume stakeholder understanding in this area, and metrics are hardcoded for each domain. Our future plan will be to use monitors to build up a profile of service measures in a domain such as an airport, and then define the values behind the qualifiers dynamically.

In addition to the user prioritisation of qualifiers, our approach also uses the concept of customer satisfaction from the VOLERE specification [RR99] to measure how happy the user will be if the requirement is implemented and how unhappy they will be if it is not. Domain experts and service consumers are asked to specify their satisfaction on a scale of 1 to 5 (1 being unconcerned and 5 being extremely happy), and also their dissatisfaction (1 being unconcerned and 5 being extremely unhappy). These measures help to indicate the importance of requirements and are particularly useful when considering scalability and the potentially hundreds of requirements for domain experts to manage.

2.1.2. Requirements management and natural language processing

The next part of the process provides the domain expert with analysis tools to manipulate the service consumer expressed requirements and to add their own requirements where necessary. The aim is for the domain expert to be able to arrange and group requirements based on the needs of end-users so that they can identify and realise ULS choreographies. Given that there is the potential for a large number of requirements from several different service consumers, the domain expert will need to consolidate these by assessing them for similarity. To do this we use techniques designed to overcome incompleteness and ambiguity in natural-language requirements. Derived and adapted from work undertaken on the SeCSE project (http://www.secse-project.eu/) the process uses a 'calculate similarity' algorithm. The algorithm specifically compares two sentences for similarity by first tokenising each requirement, and then applying word sense disambiguation and term expansion (adding terms with similar meanings) to increase the accuracy of the similarity calculation.

Natural language processing and grouping of requirements alone might not be sufficient to define the boundaries of the search space for the services to be choreographed. Therefore we also extend the process with a catalogue of user task models [ZAM].

2.1.3. User task models describing workflows and service classes

In order to define the basic parameters of the search space for services more precisely, we match and combine the requirements on the choreography with user task models. User task models describe structured activities that are often executed during the interaction with a system, influenced by its contextual environment, and performed to attain goals. They include knowledge of codified workflows, for example A must happen before B, and natural language terms describing application tasks that can be associated with service classes [ZAM].

Our approach uses the CTT (ConcurTaskTrees) task modelling formalism [PS02] to represent the knowledge about user tasks. We describe domain tasks, and manually generated reusable domain-independent task models, similar to knowledge modelling in KADS [WSB93]. For example, a description of the task "go to somewhere" describes a general process of moving from a starting point to a destination, and knowledge about this task can be reused in different travel domains. Figure 2.5 depicts part of the CTT task model of one of the user task models from our prototype task model catalogue that describes the need to calculate the distance to a location [SC04]. The task model depicts both user sub-tasks such as 'enter destination' and 'submit data', and application tasks that can be undertaken by invoked services such as 'check match with known locations' and 'confirm data validity'.

The CTT models enable the semantic gap between natural language specifications and more formal choreography specifications such as BPMN to be bridged. For example, each application sub-task is decomposed and associated with one or more service class descriptions. Each user task model incorporates one or more choreographies, with temporal operators and task sequences being used to identify service roles at particular stages of choreography.

2.1.4. Choreography patterns

Finally, the prioritized quality-based requirements and user task models are then associated with choreography strategies, which are expressed in the form of patterns by the choreography designer. The service choreography patterns encapsulate different complex choreography decisions made in the presence of user requirements. Each provides a structure for associating user requirements with different possible choreographies of classes of services that, when implemented, deliver service-oriented applications with different qualities. Each pattern considers the qualities of the services to be invoked, the qualities that different choreographies offer and how, when combined, these service and choreography qualities deliver systems of different qualities. Users of CHOReOS solutions, such as the domain expert, will match the user requirements to the service choreography patterns that will inform more detailed choreography design.

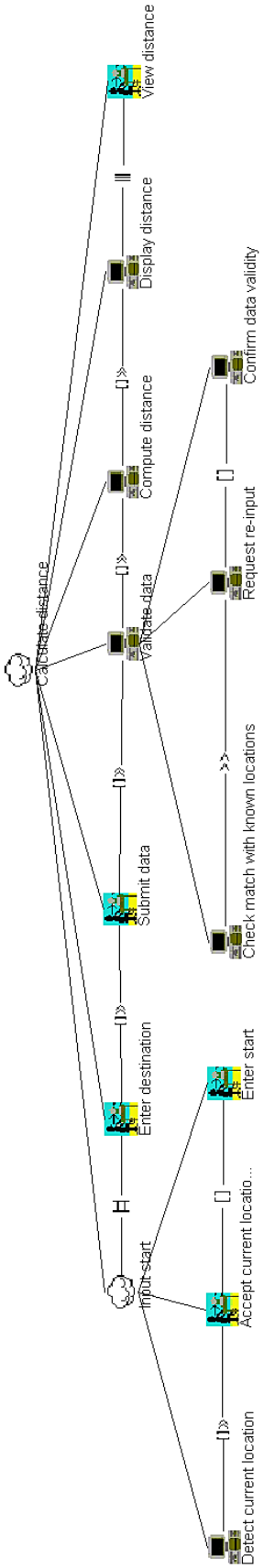


Figure 2.5: Calculate Distance example expressed as a CTT task model

2.2. Abstraction-Oriented Organization & Discovery of (Business) Services

Once the specified CTT model(s) and choreography pattern(s) are transformed into a choreography model, the business services acting as participants for the modeled choreography have to be discovered. At this point, the *eXtensible Service Discovery service* (XSD) of the CHOReOS middleware comes into play. More specifically, the discovery of business services that can act as participants of the choreography is enabled by the *Abstractions-Oriented Service Base* (AoSBM) of the CHOReOS XSD. Hereafter, we focus on the specification of AoSBM, which is developed as part of WP2, while the overall specification of XSD is provided in the CHOReOS deliverable D3.1 [CHO11a].

Prior to providing the specification of AoSBM, we discuss the FI challenges that impact on the functionality offered by the AoSBM and the way in which they can be addressed. Towards addressing these challenges, several requirements emerge that constitute the key drivers for determining the AoSBM functionality.

According to the CHOReOS deliverable D1.2 [CHO11c], the main FI challenges that CHOReOS concentrates on are *scalability*, *heterogeneity*, *mobility* and *adaptability*. Heterogeneity and mobility imply that the AoSBM should be able to gather information about available services from different kinds of sources (registries/portals/directories) by using different kinds of service discovery protocols. To deal with these issues, the AoSBM relies on the multi-protocol support of the XSD that is also detailed in D3.1.

The scalability challenge implies that the AoSBM should deal with the increasing number of business services that are published in the various sources. Many of these services may provide common/similar functionality while being heterogeneous from other perspectives (e.g. interface, non-functional properties). These services constitute *alternative* options that can play the role of a participant. Consequently, the AoSBM should provide means that allow to easily discover such alternative options. Regarding adaptability, AoSBM should provide means that facilitate the interchangeable use of alternative services in the choreography. To deal with these issues, AoSBM is based on the idea of organizing similar services into groups.

In detail, the approach forms groups of alternative services reconciling their heterogeneity and exploits these groups to provide service discovery and service browsing facilities. However, the idea of organizing similar services into groups is not proposed for the first time. As discussed in more detail in D1.1 [Tea10], several approaches have been already proposed towards the organization of services in groups/categories (or in hierarchies of categories). On the one hand, we have *top-down* approaches which assume a predefined, possibly hierarchical, organization of groups/categories. These groups may be further associated with certain semantic concepts, i.e. their definitions may rely on standard semantic description languages (e.g. OWL-S[1], SAWSDL[2] and WSDL-S[3]). Then, the services are registered in appropriate groups, either manually of semi-automatically. A typical example of a top-down approach is the one followed in the *SOA4All* project[4]. On the other hand, we have *bottom-up* approaches [Tea10], which start from service descriptions that are available and automatically group them using clustering.

In the context of the FI, the top-down organization of services is not practical since it assumes a predefined organization of groups/categories. For that reason in the AoSBM we follow a bottom-up approach. Nevertheless, our goal is not only to facilitate the identification of similar services but also to enable the identification of mappings between the similar functionalities offered by these services. In that way, the AoSBM facilitates the interchangeable use of alternative services in a choreography. Consequently, instead of using a typical clustering approach such as the ones that have been proposed

[1] http://www.w3.org/Submission/OWL-S/
[2] http://www.w3.org/2002/ws/sawsdl/spec/
[3] http://www.w3.org/Submission/WSDL-S/
[4] http://www.soa4all.eu/

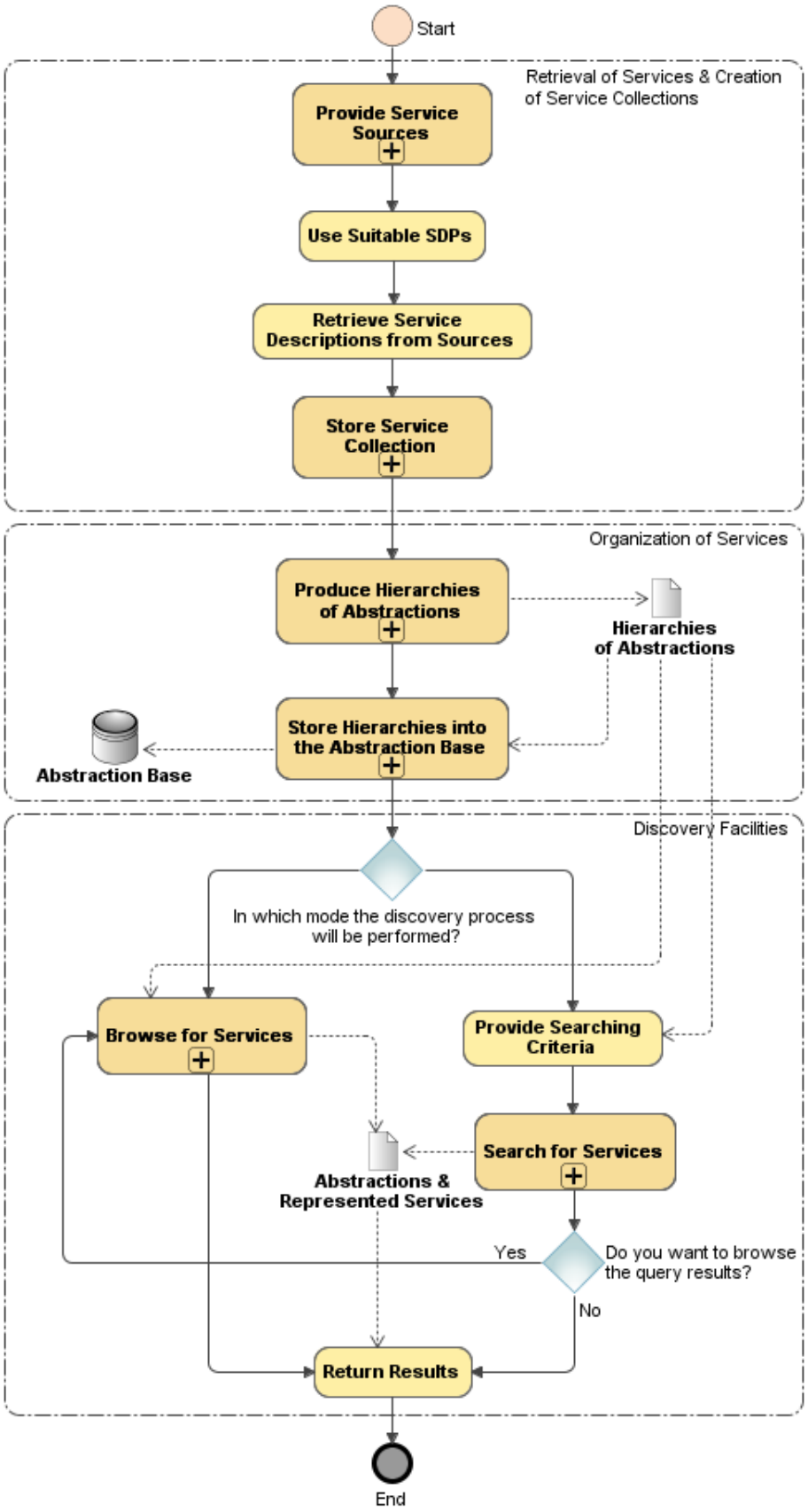


Figure 2.6: Process model of the Abstraction-oriented Organization & Discovery of Services.

in the past we propose new clustering techniques, tailored to the FI challenges. Briefly, the AoSBM organizes descriptions of services that are gathered from various sources with respect to hierarchically structured abstractions which are constructed in a bottom-up fashion. We distinguish between functional and non-functional abstractions. A functional abstraction groups alternative services that provide common/similar functionalities and is further characterized by a unified abstract interface that represents these common/similar functionalities. Moreover, the functional abstraction is characterized by mappings between the abstract interface and the interfaces of the alternative services. Similarly, a non-functional abstraction groups alternative services that provide similar non-functional properties.

An overview of the functionality offered by the AoSBM is depicted in Figure 2.6. This figure unfolds the activities that are included in the collapsed activity `Discovery Of Service` of Figure 2.2. The first part of the model comprises the activities that realize the gathering of service descriptions from various sources, using corresponding SDPs. In our case study scenario, information about hotel booking services may be obtained from popular service portals, while information about taxi public transportation services may be obtained using a SDP that operates in the airport area.

The second part of the model consists of the main activities that realize the organization of services with respect to functional/non-functional abstractions. In our scenario, alternative hotel booking services that provide similar functional/non-functional properties are organized into groups represented by corresponding abstractions. Moreover, alternative taxi services or public transportation services are also organized into groups represented by corresponding abstractions.

The third part of the model comprises the service discovery activities. The searching criteria concern the functional and/or the non-functional properties of the desired services. For the functional ones, the (domain-expert/design/developer) specifies a description of the functionality that the desired services should provide (e.g. search information about hotels), the input data required for its initiation (e.g. the arrival date and the duration of the accommodation in the hotel) and the output data of its execution (e.g. the availability of the rooms and the renting price of the found hotels). The non-functional ones concern values on the quality properties (e.g. reliability) that characterize the desired services. Having configured the service demand, the AoSBM performs the searching process over the hierarchies of functional and non-functional abstractions that organize the services. The process is based on the provided searching criteria and finally, the abstractions that satisfy the criteria along with their represented services are returned.

Following the choreography is developed with respect to abstractions (i.e. based on abstract interfaces that can be mapped to the interfaces of the alternative services).

2.3. Choreography Synthesis

A choreography is a globally-specified collaborative process among participant services. The interaction defined by the choreography has to be projected (in a peer-style fashion) among the different participants according to the “local” roles they play. In other words, as detailed in the deliverable D1.3 [CHO11d], a choreography specification can be projected to the different roles played by the CHOReOS components participating to the specified choreography. In particular, for automated synthesis purposes, in [CHO11d], the peer-style specification is derived as an LTS-based specification.

Choreography synthesis concerns the realization of distributed *coordination delegates* that, on top of the CHOReOS middleware, cooperatively work to support the enactment and execution of the choreography. By relying on the facilities offered by the SBM, M2M techniques [HMY06] are used to refine the *Choreography Model* together with the service *Behavior Protocol Automata* into a *Peer-style Specification* (see Figure 2.7). Service behavioral models are automatically synthesized by the *Synthesis of Behavior Protocols* activity starting from the interface descriptions of services that have been discovered within the *Abstraction Base* (see Section 2.2). This activity is based on a combination of syntactic interface analysis and testing [BIPT09].

The synthesis process is able to understand how to actually coordinate the discovered services to

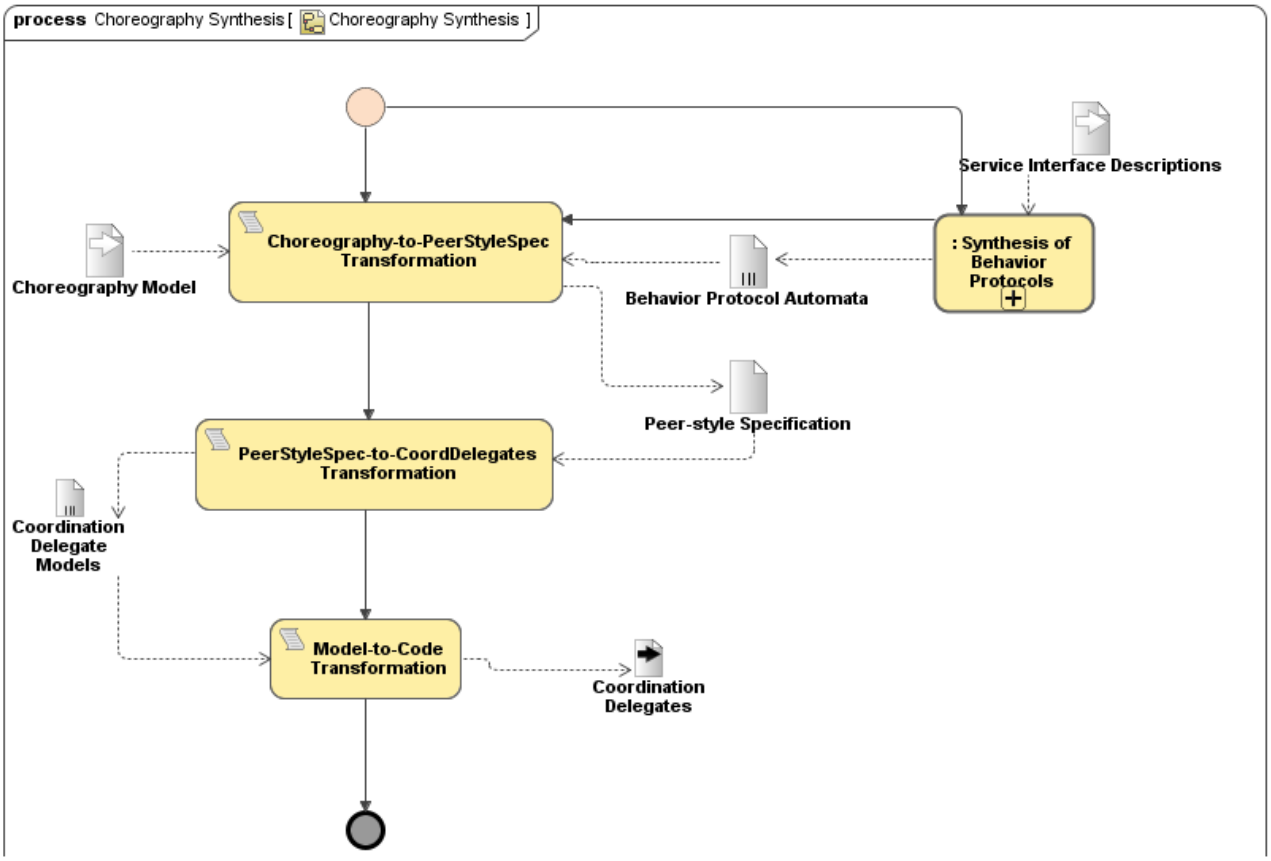


Figure 2.7: Choreography synthesis

suitably realize the specified choreography. For instance, it might be the case that the discovered services, although potentially suitable in isolation, when interacting together can lead to, e.g., concurrency and interaction mismatches, safety and timeliness violations. Thus, although a given set of services, if coordinated in the right way, can be used to achieve the specified choreography, they can completely miss the choreography's goal if coordinated in a different way. Further applying M2M transformation, the synthesis method will produce abstract coordination delegate models. Such models, accounting for the services' functional and non-functional abstractions, force the collaboration of the discovered services to guarantee the specified choreography. To actually realize the choreography, the abstract coordination delegate models are then concretized into actual software artifacts by means of M2C transformation. The generated coordination delegates suitably access and coordinate the discovered services by relying on the communication facilities provided by the CHOReOS middleware (see Figure 2.1 and Figure 2.2).

2.4. Deployment and run-time of ULS choreographies

The CHOReOS middleware must be capable of providing the required runtime support to deploy, enact, monitor, and dynamically reconfigure large-scale choreographies. To accommodate the choreographies large-scale aspects, the CHOReOS middleware relies on Cloud and Grid Computing. Cloud Computing is the default mechanism for providing scalability while Grid Computing is used in more specific cases in which intensive parallel computation is required.

The middleware Cloud Computing features are implemented by the Node Pool Manager component, which is able to allocate new nodes (virtual machines) in multiple underlying execution platforms. Such underlying platforms can be public clouds such as Amazon EC2, HP Open Cirrus or private clouds systems, like the Open Nebula open source cloud middleware.

The Service Deployer component of the CHOReOS middleware is responsible for allocating new CHOReOS nodes from the Node Pool Manager according the choreography specification generated by the synthesis process. In these nodes, CHOReOS executes the coordination delegates, which are the major components for service access at runtime in CHOReOS infrastructure. The coordination delegates deployed on the cloud nodes are executed on top of the Petals DSB (Distributed Service Bus) and use the bus for communication with other services.

The described process of deployment of a coordination delegate is pictured in the Figure 2.8. For more details about the CHOReOS execution runtime, see the deliverable D3.1, the CHOReOS Middleware Specification.

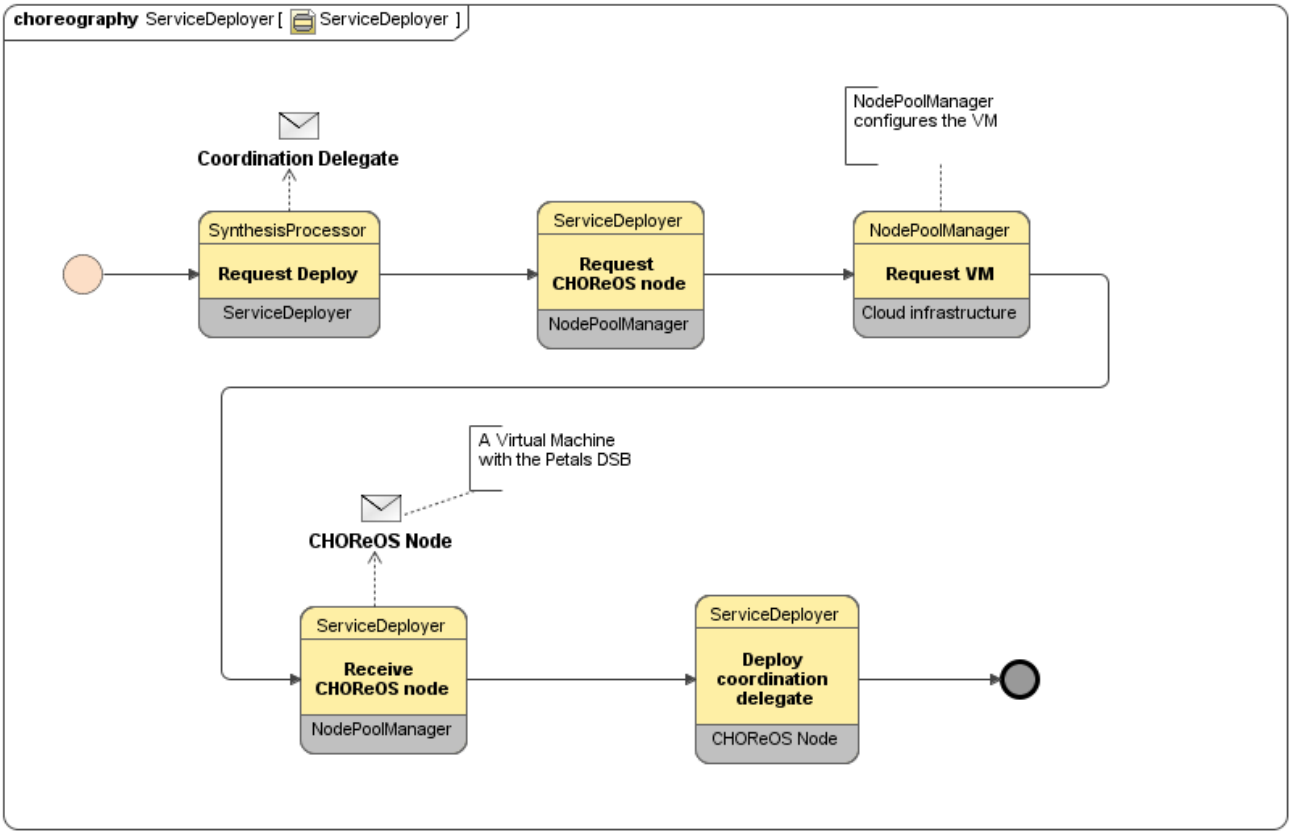


Figure 2.8: Coordination delegate deployment process

2.5. Governance, V&V and Monitoring

SOA governance is usually meant as a set of best practices and policies for ensuring interoperability, adequacy, and reuse of services over several different platforms. Both design time and runtime governance are considered in SOA governance solutions.

Achieving SOA governance is a complex and articulated task that usually involve many different and orthogonal domains (i.e., technical, legal, standard procedures, internal procedures). Nevertheless, in the most abstract formulation, SOA governance is usually realized through a cycle consisting of: policy definition, auditing and monitoring, and finally evaluation and validation. The expected good behavior is expressed through the definition of both models and policies.

Within the CHOReOS development process, it is crucial identify the roles that people assume with respect to a framework for SOA governance and V&V. Specifically, as people formulate decisions in terms of policies (e.g., design policies, development policies, run-time policies) that would be audited, monitored, and validated, it is necessary to identify the roles and the responsibilities of the actors

dealing with Governance V&V, and Monitoring

As deeply argued in Deliverable D4.1 [BAP11], the CHOReOS Governance Framework distinguishes between the following roles: *Governance Manager*, *V&V Manager*, *Choreography Designer*, *Service Provider*, and *Service Consumer*. Briefly, the Governance Manager governs the stages for service lifecycle and participates in defining policies, the V&V Manager is responsible for the definition and the configuration of the V&V strategy; the Choreography Designer is the responsible of a given choreography specification, the Service Provider roughly represents both the developer, and the maintainer of a set of services; while the Service Consumer discovers and then uses services form a repository/registry.

Basing on such roles, CHOReOS identifies different scenarios for both Governance V&V, and Monitoring activities. In the following, we summarize such scenarios; nevertheless, both their overall description, and their discussion is left to [BAP11].

The first scenario for Governance V&V considered by the project is referred as **Choreography Registration Use Case**. Specifically, when the design of a choreography is completed, the Choreography Designer can make it available by registering a specification of such choreography on a dedicated registry (e.g. the Governance Registry [BAP11]). From such registration the lifecycle at run-time of the choreography is regulated by means of a set of policies that are mainly defined and agreed between the Governance Manager, and the V&V Manager. For example a first policy investigated within [BAP11] concerns the rules under which a choreography is enactable, another deals with the criteria and the parameters for evaluating the quality associated with of a choreography.

The second Governance V&V scenario concerns the **Service Registration Use Case**. According to the CHOReOS architectural style [CHO11d], three different scenarios are described concerning the ways in which a choreography can be composed, which are “ad-hoc”, “role-based” and “requirement-based” (see [CHO11d], Chapter 5). The role-based scenario, in particular, foresees that Service Providers promote their services (i.e. either develop, or adapt) as participants “fitting” one or more roles to be played in a given choreography. This scenario is conceived to fit the case that a choreography has already been enacted, registered in the Governance Registry [BAP11], and in the dynamic FI world, services can dynamically enter and exit the choreography, playing one of the specified roles. The same service can play different roles in different choreographies depending of the provided functionality. In other words, such scenario foresees that upon registration (or otherwise by updating a previous registration) in the Governance Registry, the Service Provider specifies some role(s) the service will play and in which choreography(ies). From a testing perspective, this is the most challenging case, as the Choreography Designer needs to check whether the service can comply to the role, by testing for conformance to the specification.

In this sense, the registration of a service results as a critical point for the Governance Manager, and the Choreography Designer that wants to guarantee the registered services abide by both the functional and non-functional specification foreseen by a choreography. As an extension of the work proposed in [BP05], the Governance framework we are developing in CHOReOS supports V&V activities that aim at testing if a service implementation actually conforms to the role that the Service Provider claimed during its registration. Furthermore, in this scenario, it is equally important for the service registry to re-test (i.e. either periodically, or event-driven) a service that has been already registered in order to guarantee that its behaviour did not change over time.

A common practice in SOA, is that Service Consumer may negotiate the application of both specific SLAs, and specific policies interacting with the services offered by a given Service Provider. Furthermore, in a more extensive scenario, SLA as well as policies could be requested at the choreography level directly by Choreography Designer. Thus the third scenario for Governance V&V is referred as **Service Monitoring Use Case**, and it describes how to provide support for observing if such agreements or policies are actually honored. In this use case, each of the roles listed above should be able to define monitoring rules that focus in analyzing the events related to lifecycle of a service, reporting any violation of the negotiated behaviour that might occur. Also [BAP11] describes some specific policies that could be applied reacting to a violation: for example deleting a service from the registry, or decreasing its quality rating.

Finally, a last Governance V&V scenario is the **Choreography Monitoring Use Case**. Specifically, in addition to the scenario where the constrains are associated to a service, also a service choreography may undergo to SLAs and policies that concern the interactions it specifies as a whole. In this use case, all the role described above, but especially the V&V Manager, and the Choreography Designer, may be interested in defining monitoring rules that observe how each enactment of a choreography is proceeding.

2.6. Case Study: Passenger-friendly Airport.

The following case study will be considered in next chapters to explain two building blocks of the development process model: the domain expert requirements specification framework (Chapter 3) and the large scale service base management (Chapter 4).

Building on the description of the WP6 use case, we consider a simple scenario that describes the sequence of activities of an airline passenger. While in the airport, the passenger is notified by the airline of a cancellation to her flight. The airline informs the passenger that the flight has been rescheduled for the day after. Furthermore, it notifies interested passengers that free bus transportation services are available for dispatching passengers to selected hotels. Each hotel is chosen, among a set of hotels operating with the airline, according to the passenger preferences. In alternative, for those passengers that do not need a hotel, the airline makes available a taxi service to reach a specified destination.

In such a scenario, the employment of choreographies can play an important role to coordinate the various actors and services involved from the time the cancellation is announced until it is time to embark the flight. Note that, this scenario may include two different options: *bus-to-hotel* or *taxi-to-destination*. Furthermore, according to the passenger preferences, within one of the two options mentioned above, different services can be selected as choreography participants, e.g., if the passenger prefers to stay in a peaceful environment away from the downtown, then the synthesized choreography will involve a bus company service and a hotel booking service different from the ones that would be selected for staying in a hotel in downtown.

The choreography participants can be the networked services provided by the airline company, the ones deployed on the CHOReOS-enabled passenger's smart phone, and either the hotel and bus booking services or taxi services that have been discovered so as to "match" the service roles in the choreography.

3 Domain Expert Requirements Specification Framework

In this chapter, the process for the domain expert specification outlined in Section 2.1 is further detailed in the form of a framework for expressing requirements and producing a first draft choreography specification. Figure 3.1 shows a BPMN2 process diagram specifying the flow of the requirements specification development process activities and the manipulated data. There are 4 main process activities: 1) specify requirements in mobile service consumer tools; 2) analyse requirements (undertaken by the domain expert); 3) execute similarity algorithm and group requirements; and 4) match requirements to task models. The inputs to the overall process are the service consumer requirements, of which there can be many service consumers with many requirements. Other inputs to the process include an ontology for non-functional requirements and service quality measures, user task models and choreography patterns. The final output from the requirements specification process is a first draft choreography specification (in BPMN2 language) which serves as input to the next phase of the overall CHOReOS dynamic development process.

3.1. Specify requirements in mobile service consumer tools

The requirements specification process begins with the service consumer who specifies requirements, or user needs, using a structured approach that includes qualitative qualifiers, satisfaction ratings and prioritisation. This aspect of the process model is instantiated by mobile service consumer tools, in line with the Future Internet definition of mobility described in D1.2 [CHO11b], as services are increasingly accessed by mobile devices rather than fixed line connections. In particular, an iPhone app (application) has been developed for the CHOReOS project to instantiate this part of the process. The user interface of the app is presented on the left-hand side of Figure 3.2.

The service consumer begins by entering their service need in natural language into the textbox. In this example the app is configured for the airport domain and the user is requiring services associated with travel and accommodation. The predictive text auto-completion function for the service description can be extended in order to enforce structure through a set domain specific terms e.g. departure, arrival, gate, security etc. The service consumer then expresses which qualities are most important to them using the sliders, and thus providing an order of priority to the service qualities. These qualities refer to: speed, precision, robustness, usability and security. The magnitude of the importance for each quality is recorded on a scale of 1 to 5. Constraints applied to the sliders mean that the user can only express the highest priority for one quality. Finally, the service consumer is able to express their satisfaction if the service need is met and their dissatisfaction if it is not, using the bottom two sliders. Again, these attributes are recorded on a scale of 1 to 5. Having expressed their service needs, the service consumer can then send their query and receive an email confirmation of their request. The expressed attributes are recorded in an on-line MySQL database, shown on the right-hand side of Figure 3.2, along with the attributes of longitude, latitude and time. These situational attributes add context to the service needs and can be used by the domain expert as part of their analysis of the requirements.

The data presented in Figure 3.2 refers to the passenger-friendly scenario introduced earlier. The service consumers, i.e. the passengers, express their service related needs throughout time during

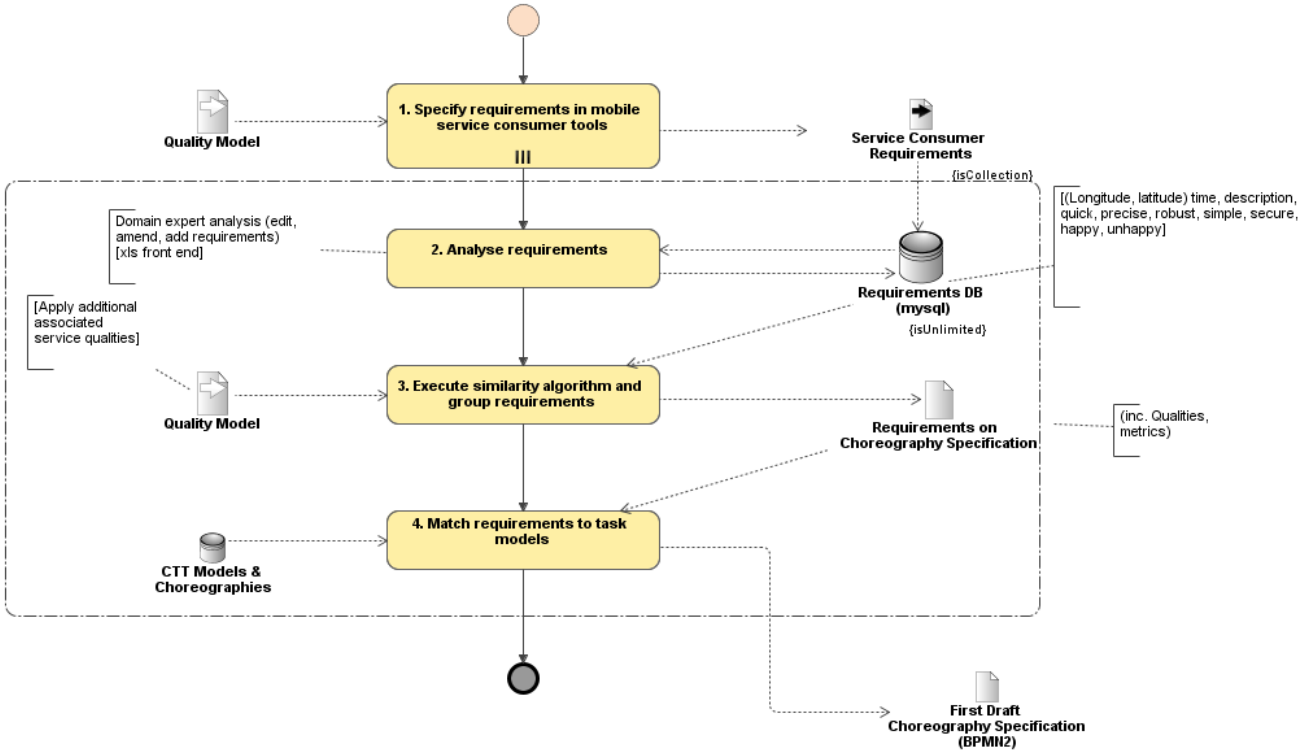


Figure 3.1: Activities and the manipulated data of the requirements specification development

normal airport operations. Also, of particular interest to the domain expert in this scenario, are the requirements expressed during periods of unforeseen circumstances, such as flight cancellations. These requirements can be analysed by the domain expert when they are considering a new service to be choreographed to handle such scenarios. In this example, the passenger wants to be informed of all the hotel options near to the airport. They specify qualities such as wanting the information quickly, but the exact precision of the hotel locations are less important in this scenario as they will be transported there by the bus service. The passenger also expresses their satisfaction if the service is provided (they would be happy) and their dissatisfaction if it is not (they would be very unhappy). From this, the domain expert is able to formulate an overall requirements specification for services to be choreographed.

Associated with the expression of service consumer requirements is a quality model which relates the user requirements on service-based applications to QoS on services aggregated in these applications. This model represents the quality ontology for the CHOReOS project.

The CHOReOS quality model, developed from research undertaken for the SMI model of cloud service characteristics [ZLHM11], was described using a tailored form of the *i** modelling approach [YM94]. *i** was originally conceived as a goal-based technique for modelling information systems composed of heterogeneous actors. These actors are expected to have different, often competing goals and depend on each other to undertake their tasks to achieve these goals [YM94]. *i** has been used successfully to model service-centric information systems (e.g. [LKP+08]), showing relationships between service behaviours and requirements, and trade-offs between the achievement of service qualities. This is achieved by modelling soft goals, that describe qualities that can be achieved to a greater or lesser extent, equivalent to many types of service characteristics. The soft goals are related using contributes-to links, which show how the achievement of one soft goal can contribute positively or negatively to achieving another.

The CHOReOS quality model is presented in Figure 3.3. Starting from the left-hand side of the figure, the service consumer expresses their user requirements and associated qualities as described above using the iPhone app. The prioritised set of 5 qualities is then related to qualities on the overall

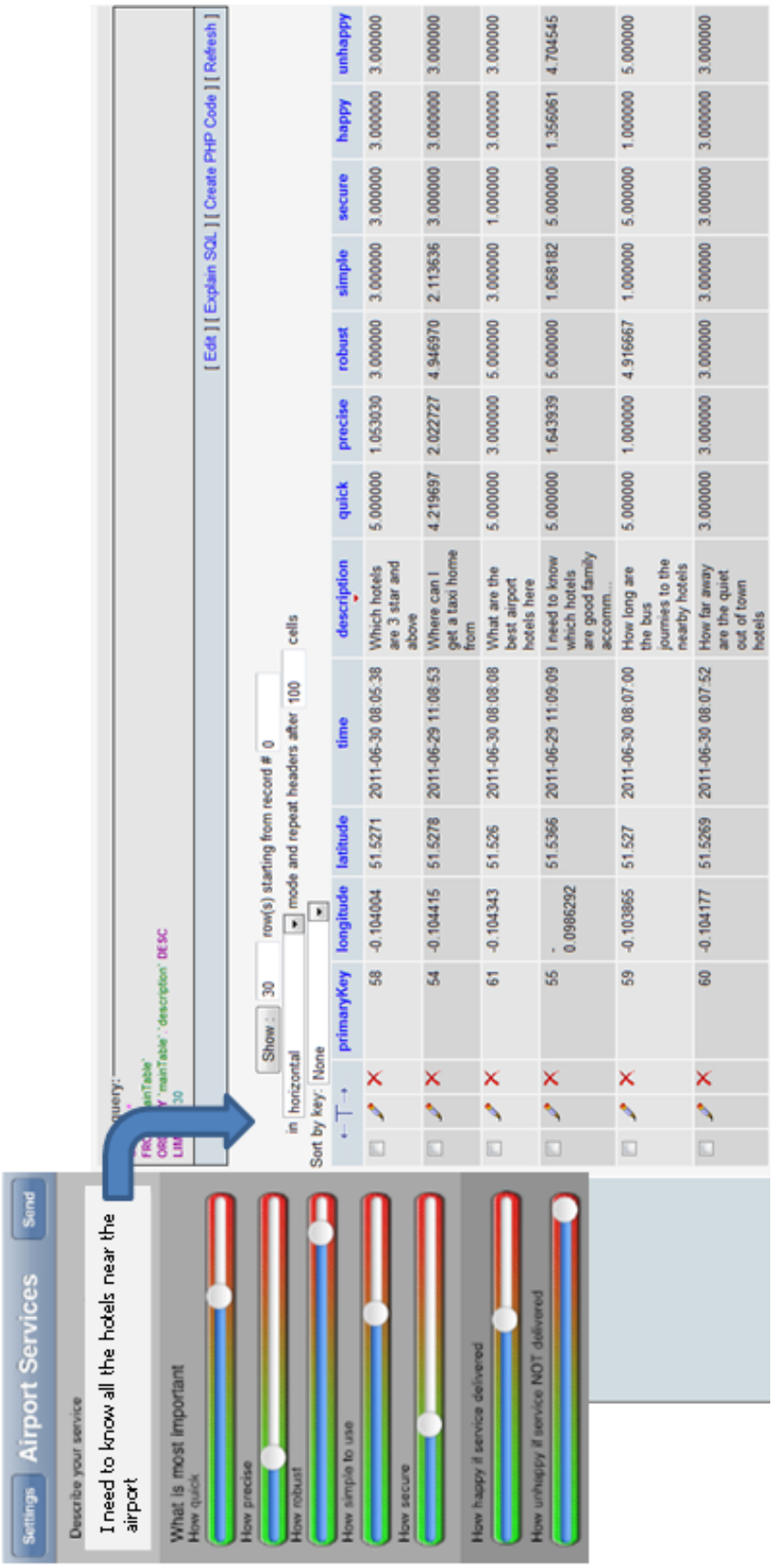


Figure 3.2: Service consumer expression of requirements using the CHOReOS iPhone app

service-based application. For example, the user response to 'how robust' is related to the dependability quality. From this, the model decomposes the higher-level qualities to derive lower-level non-functional requirements on the services and the relationships between them. For example, it can be seen that maintaining service response time can contribute positively to achieving usability, and achieving capacity can contribute positively to maintaining throughput.

A standard template is used to describe each characteristic. The template starts with a more complete definition of the characteristic, and then describes each quantitative measure associated with the characteristic. For each measure one or more metrics are specified along with a defined data source. Table 3.1 reports the template for performance, one of the 5 main qualities expressed by the service consumer. Three metrics are specified for the quality - the *mean rate of invocation*, the *mean utilization of the CPU*, and the *upstream latency standard deviation*. The first is computed using the arithmetic mean of the number of invocations in a particular time frame. The second is computed using the CPU utilization over time, and the third is the standard deviation of experienced upstream network latency over some period. A set of 31 qualities have been defined in a similar fashion, and are available in Appendix A.

Quality	Performance		
Description	The performance of a web service represents how fast a service request can be completed. It can be measured in terms of throughput, response time, latency, execution time, and transaction time, and so on. In general, high quality web services should provide higher throughput, faster response time, lower latency, lower execution time, and faster transaction time.		
Measure	**Name**	MeanInvocationRate	
	Comments		
	Description	The arithmetic mean of the number of invocations in a particular time frame.	
	Metric	**XSD Type**	float
		Unit	Units of frequency
		Value Range	0.0 – ∞
		Source	
		Calculation Algorithm	
		Coding	
Measure	**Name**	MeanCPUUtilization	
	Comments		
	Description	The arithmetic mean of CPU utilization over time and potentially over a number of systems in a cluster.	
	Metric	**XSD Type**	float
		Unit	percent
		Value Range	0.0 – 100.0 inclusive
		Source	
		Calculation Algorithm	
		Coding	
Measure	**Name**	UpstreamLatencyStandardDeviation	
	Comments		
	Description	The standard deviation of experienced upstream network latency over some period. This metric assumes that latency varies randomly.	
	Metric	**XSD Type**	float
		Unit	Units of time
		Value Range	0.0 – ∞
		Source	
		Calculation Algorithm	
		Coding	

Table 3.1: Definitions, measures and metrics for the service performance quality

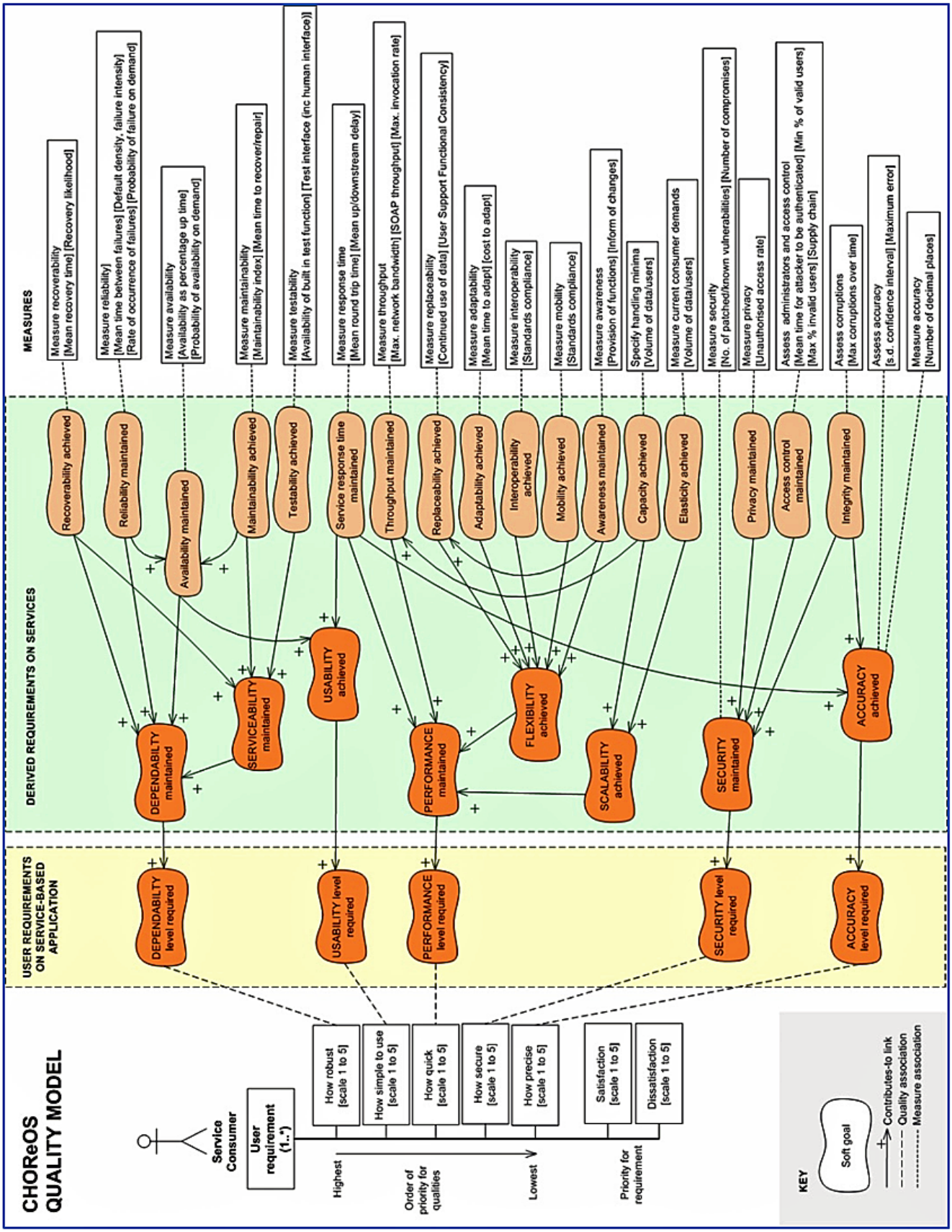


Figure 3.3: CHOReOS quality model showing user requirements on the service-based application, derived requirements on services and quality measures

3.2. Analyse requirements

The second stage of the process concerns the domain expert analysis of the requirements and is implemented in an MS Excel database front-end developed for the CHOReOS project. The Excel-based tool accesses the MySQL database and provides analysis functions for the domain expert to manipulate the data. Existing service consumer requirements can be edited and amended, and the domain expert can also add their own requirements using their domain knowledge. Edited and added requirements are flagged in the database, and the domain expert can also flag service consumer requirements that have been reviewed and accepted. The domain analyst is able to generate different views of the data set by filtering the requirements. Filtering includes selecting the last 100 requirements entered into the database (based on time stamps, this could be any number the domain expert desires), those with particular qualities (e.g. those with high-level security, high performance, high customer dissatisfaction score etc.) These functions are provided to help the domain expert to pull out individual requirements in order to form a set of requirements for identifying and specifying choreographies.

This part of the process requires analyst input and is therefore a manual task. For example, the domain expert will need to interpret the different qualities associated with the requirements provided by different service consumers. Conflicting preferences and priorities may indicate different services required by different user types. Also, similar requirements may need to be consolidated. To assist the domain expert in doing this they are provided with a similarity function, as described in the next section.

3.3. Execute similarity algorithm and group requirements

In the next stage of the process, the domain expert can use a 'calculate similarity' algorithm to help organise and group requirements - for example, to remove duplicates and to identify the correct level of abstraction for similar requirements. In addition, the CHOReOS quality model is used at this stage to apply the derived quality requirements on the services to be choreographed.

The WordNet on-line lexicon fulfils an important role in this algorithm by providing word senses and definitions which disambiguate terms. It is also used to add semantic relations between terms with similar meanings to make the similarity calculation more accurate. WordNet is a lexical database inspired by current psycholinguistic theories of human lexical memory [MBF+90]. It has two important features. First, it divides the lexicon into four categories: nouns, verbs, adjectives and adverbs. Word meanings, called senses, for each category are organized into synonym sets (synsets) that represent concepts, and each synset is followed by its definition or gloss that contains a defining phrase, an optional comment and one or more examples. Second, WordNet is structured using semantic relations between word meanings that link concepts. Relationships between concepts such as hypernym and hyponym relations are represented as semantic pointers between related concepts [MBF+90]. A hypernym is a generic term used to designate a whole class of specific instances. For example, vehicle denotes all the things that are separately denoted by the words train, chariot, dogsled, airplane, and automobile, and is therefore a hypernym of each of those words. On the other hand, a hyponym is a specific term used to designate a member of a class, e.g. chauffeur, taxidriver and motorist are all hyponyms of driver.

The algorithm has 4 key components. The first is natural language pre-processing, which initially tokenises the text strings using white space as the delimiter for splitting the sentence. In the second step, the algorithm identifies complex nominals (e.g. the term boarding gate) based on domain-specific terms defined within a glossary and term definitions in WordNet. In the third step the algorithm identifies and removes all terms defined in a list of stop words (e.g. prepositions and pronouns). Next, all terms are tagged with their corresponding part-of-speech (e.g. singular common noun, comparative adjective, etc.) and classified accordingly using an improved version of the Brill Tagger [Bri92]. In the fifth step each term is converted to its morphological root (e.g. travelling to travel). Finally, all duplicate occurrences of a term are removed so that each term is stored only once with its cardinality, as reported in [WS03]. For the second component, the algorithm applies procedures to disambiguate each term

by defining its correct sense and tagging it with that sense (e.g. defining security as airport security and not a type of financial asset). Assigning the correct sense to a word in context requires syntactic, semantic and pragmatic knowledge about the word itself, its part of speech, and its context [SW01]. With this component the algorithm disambiguates the text by applying a combination of procedures such as defining synonyms and hypernyms, and using frequency-based senses i.e. assigning the most frequent sense to a term irrespective of its context. The third part of the algorithm expands each term with other terms that have similar meaning according to the tagged sense, to make it more complete and increase the likelihood of an accurate similarity score (e.g. the term traveller is synonymous with the term passenger). The algorithm only considers disambiguated terms to expand, as expansion of incorrect senses is not likely to improve the accuracy of the result and would be inefficient. Finally, the fourth component of the algorithm matches the expanded and sense-tagged terms of one text string to another. The result is provided on a scale of 0 to 1.

The calculate similarity web service is called from the Excel tool by posting a SOAP 1.1 envelope via an HTTP request written in VBA (Visual Basic for Applications) code. SOAP (Simple Object Access Protocol) is an XML based protocol comprising three parts: an envelope that describes what is in the message and how to process it, a set of encoding rules for expressing instances of application-defined data types, and a convention for representing remote procedure calls and responses (further information available at http://www.w3.org/TR/2000/NOTE-SOAP-20000508/). SOAP can be used in combination with HTTP, as coded in the VBA module in the CHOReOS Excel-based tool. The web service, along with the SOAP 1.1 request and response code, is shown in Figure 3.4.

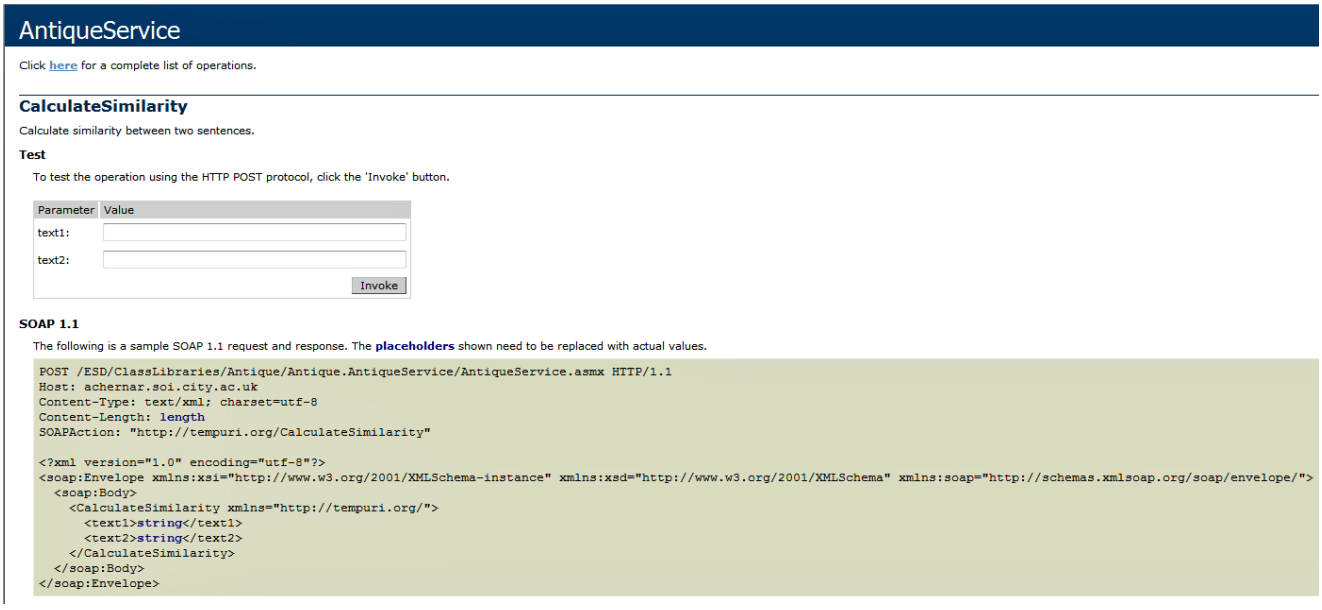


Figure 3.4: CalculateSimilarity Web-service showing the SOAP 1.1 request & response code

The calculate similarity web service is called from the Excel tool by posting a SOAP 1.1 envelope via an HTTP request written in VBA (Visual Basic for Applications) code. The web service, along with the SOAP 1.1 request and response code, is shown in Figure 3.5. The clustering functionality provided by the CHOReOS domain expert tool is shown in Figure 3.5. This function takes a set of service consumer requirements identified by the domain expert as input and provides the following features from which the data set can be manipulated: (i) the domain expert can move requirements from the original data set across to a clustered requirement group using the right arrow button. The requirement is removed from the original set in this view. A requirement from the group can also be moved back into the main set if the analyst changes their decision; (ii) the domain expert can select a requirement from the original set and run the calculate similarity algorithm, which processes the similarity between the selected requirement and all the others in the set; (iii) the domain expert can then use the re-ordered requirements, based on similarity, to move relevant requirements across to the group. It is expected that a score of around

0.35 and above would indicate a significant similarity between requirements; and finally, (iv) the domain expert can create the clustered requirements group and commit it to the database. The output from this process is set of grouped requirements for identifying and specifying choreographies for a particular scenario.

Referring back to the passenger-friendly scenario example, Figure 3.5 shows the similarity scores generated from the requirement 'I need to know all the hotels near the airport'. The requirement with the most similar match is 'I need to know which hotels are good family accommodation' with a similarity score of 0.48. The analyst is then able to assess all of the requirements with the highest similarity scores and create a set of requirements, for example, the requirement 'what are the best hotels here' has already been added to the clustered requirements group.

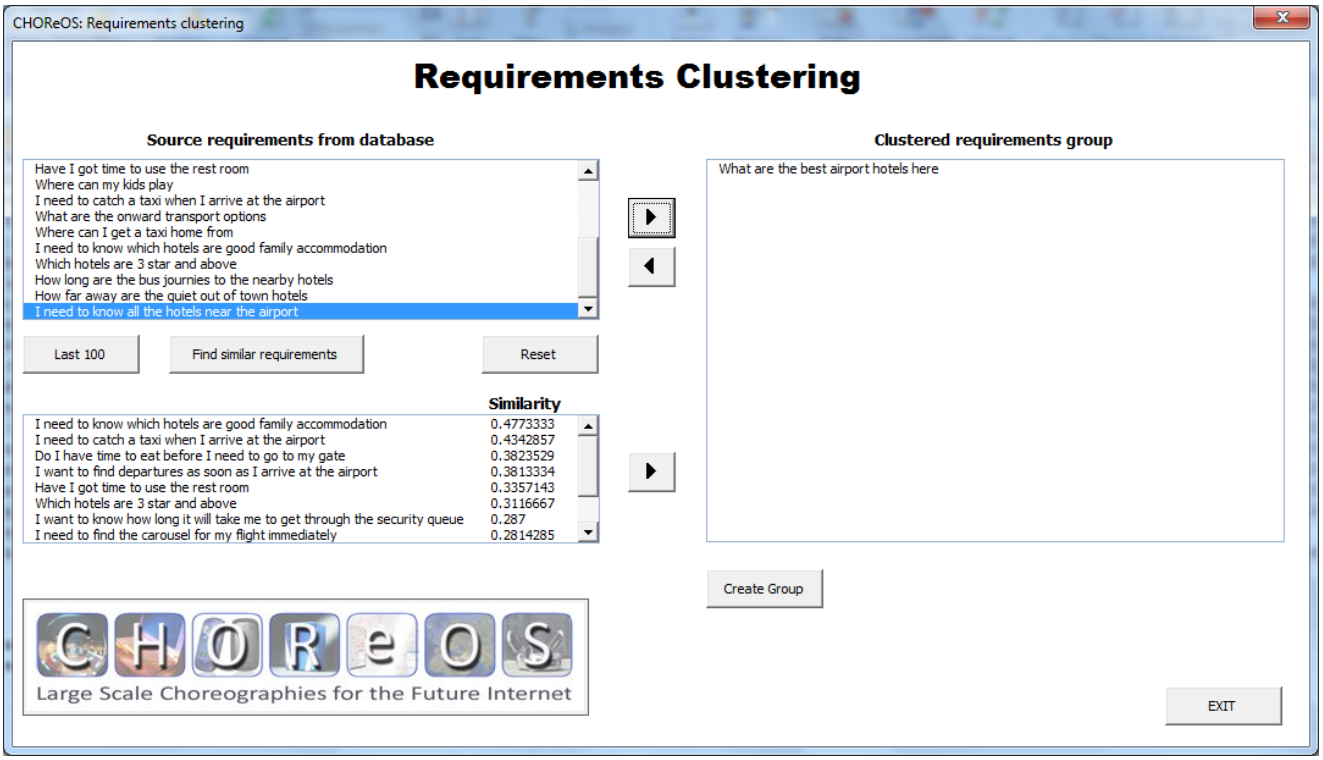


Figure 3.5: An prototype of the CHOReOS Domain Expert tool showing the clustering function

3.4. Match requirements to task models

The requirements on the choreography specification, along with an initial set of discovered candidate services from the CHOReOS service base, are matched to CTT task models using the T-EDDiE tool. These task models are being developed for the domains featured within the CHOReOS use cases, and include knowledge of codified workflows (tasks) and natural language terms describing service classes. The CTT models enable the semantic gap between natural language specifications and BPMN2 choreographies to be bridged. For example, each application sub-task will be decomposed and associated with one or more service class descriptions. Each user task model will incorporate one or more choreographies, with temporal operators and task sequences being used to identify service roles at particular stages of choreography. The T-EDDiE tool will be used to process the matching and create the choreography patterns to produce the final output a first draft choreography specification.

The requirements on the choreography specification are matched to user task models using the T-EDDiE tool [ZAM]. The requirements act as a set of terms from which relevant models from the user task model database are identified. These terms used for matching include the derived quality requirements on services that are applied from the CHOReOS quality model. The functional and non-functional

requirements are related to sub-tasks in the user task model. Furthermore, these requirements on the described service classes, or application tasks in the CTT model, are later associated with abstract characterizations of services to be discovered from the CHOReOS service base (see Chapter 4).

The task models in the CHOReOS requirements process are expressed using the CTT formalism [PS02]. Each sub-task specified in the CTT model can be: (i) an *abstract task* that is decomposed further; (ii) a *user task* undertaken by the user; (iii) an *interaction task* carried out by the interaction of a user with a software system, or: (iv) an *application task* fully undertaken by software [PMM97]. Possible sequences of these sub-task types are described by CTT operators such as *concurrent* and *enable with information passing*. Continuing our simple scenario example from earlier, the requirements in the user's problem domain provide T-EDDiE with a set of terms to parse to match to the task model catalogue. The retrieved task model *Get to accommodation* task is specified graphically as a CTT task model in Figure 3.6. For example, the matching process will have identified terms concerning finding accommodation and travelling to accommodation.

The task hierarchy is described in a tree structure using task decomposition. The higher level task *Get to accommodation* is decomposed into lower level subtasks that execute it: *Request booking*, *Process request*, *Choose alternative*, and *Confirm booking*. All four of these subtasks are themselves decomposed further for example, *Request booking* comprises *Enter accommodation preferences* and *Enter transport preferences*. Graphical syntax elements indicate each of the four tree node types. Application tasks (computer), such as *Confirm booking*, are executed by a software element without explicit intervention from users. Interaction tasks (person at a computer), such as *Request booking*, are activities that require interaction between a user and a system in order for them to occur. User tasks (a human head), such as *Decide*, are activities undertaken by the user alone. Finally, abstract tasks (cloud), such as *Choose alternative*, require complex actions and cannot neatly fall into either of the other task categories of CTT.

Relationships between tasks at a same hierarchical level and their occurrence in time are described using temporal operators. For example, *Request booking* enables and passes on information (here, the preferences elicited) to *Process request*, which in turn enables and informs *Choose alternative*. This relationship is called "enable with information passing" ($[] >>$). At this same hierarchical level, the final temporal operator ($[>$) between *Choose alternative* and *Confirm booking* indicates that the CTT is disabled as this is the end of the overall user task. At the lowest hierarchical level in this example, *Enter accommodation preferences* and *Enter transport preferences* are deemed to be independent tasks (H) as they do not require information flow from each other, and can theoretically occur in any order. The temporal operator ($|||$) designates *Find accommodation* and *Find transport* as concurrent tasks and the user task *Decide* enables ($>>$) the user to *Send* choice. Finally, choice ($[]$) is represented between *Send choice* and *(Re)request booking*, as the user can either accept one of the alternatives or reject all of them and restart the process. Choice is also expressed for the application tasks *Send booking info* or *Send failure message*.

These task sequences and associated temporal operators are important for determining the service order in BPMN2 choreography diagrams. A set of simple rules is being developed for the CHOReOS project in order to map between the CTT model and the semantics used to describe the choreography patterns. In our example, the CTT model has been translated into the BPMN2 choreography diagram shown in Figure 3.7. The overall goal, or top-level task, of the CTT model *Get to accommodation* relates to the final choreography event and the end state *Success*. Following the start of the choreography, the initial BPMN2 activity *Notify Passenger Cancelled Flight*, the user task tree then informs the order and flow of activities and messages. For example, the interaction task *Request booking* is expressed as the activity *Send Passenger Request for Bus and Hotel*, and *Process request* is shown as *Send Bus and Hotel Alternatives*. The Task model adds further decomposition to these activities to help identify the relevant service classes and identify choreography roles. The enable with information passing link in the CTT model is represented as message exchange in the choreography diagram. Other temporal operators are expressed, for example gateways can be determined by the temporal operator choice ($[]$) as reflected by the passenger determining whether any of the alternatives are satisfactory. The final

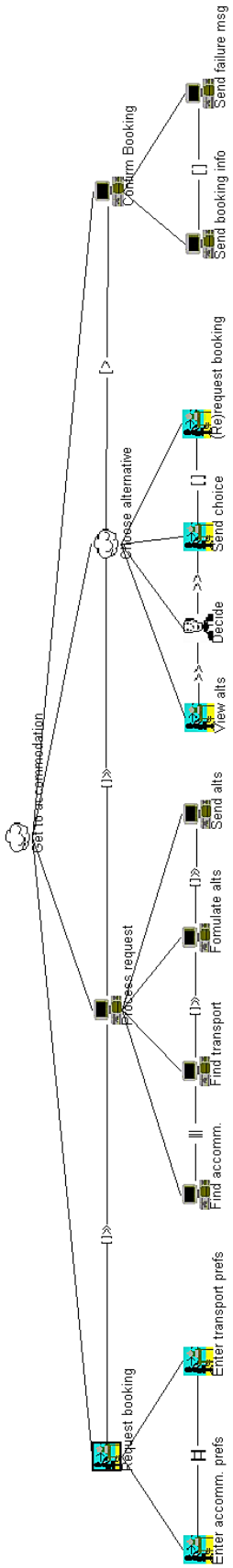


Figure 3.6: Get to Accommodation example expressed as a CTT task model

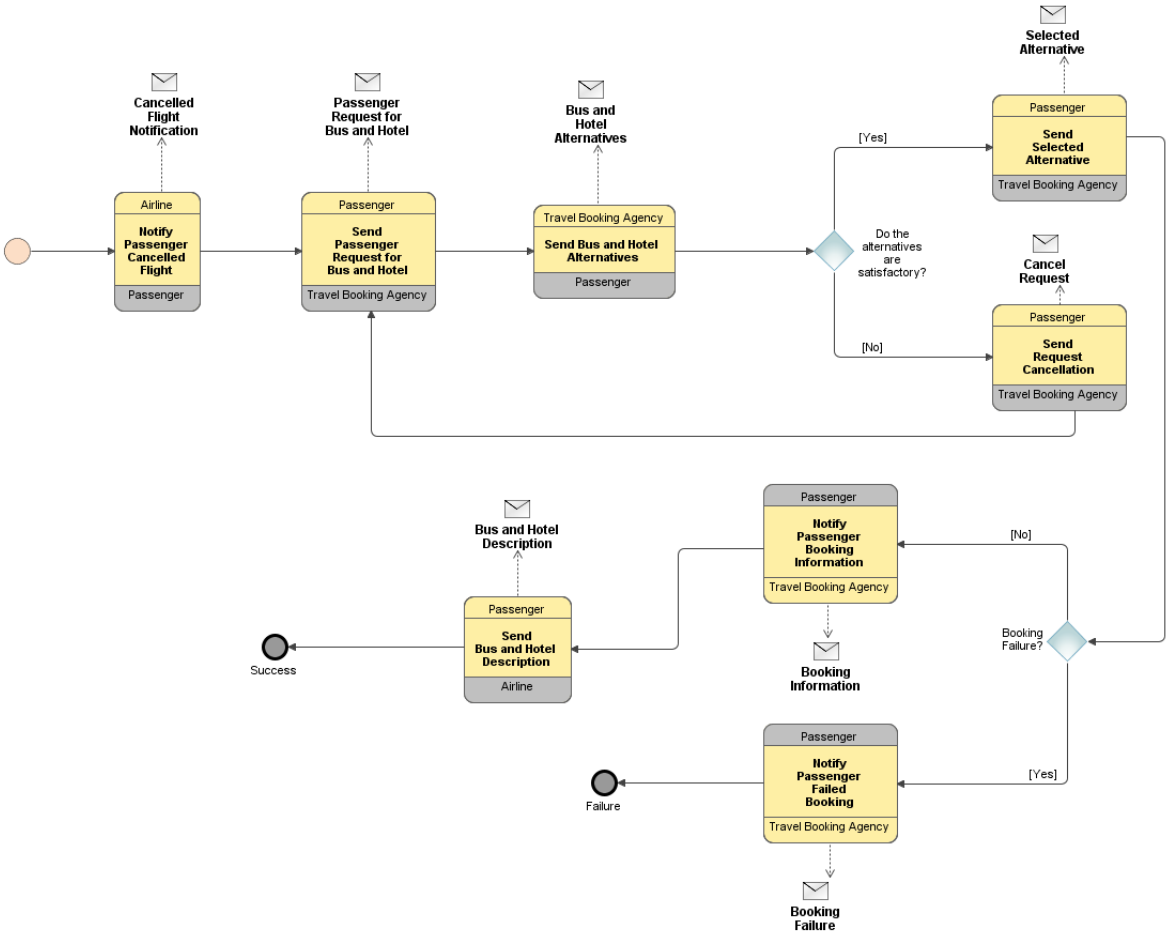


Figure 3.7: Choreography diagram

subtask can be shown as disabling the process and is reflected as a final choreography activity and an end event, in this case success or failure of the travel agency booking.

4 Abstraction-oriented Service Base Management

The primary purpose of this Chapter is to specify the AoSBM functionality from an external and internal point of view. From the external perspective, we describe the way in which the AoSBM interacts with its end-users. From the internal perspective, we supply an architectural view of the AoSBM by describing the subsystems that fulfil its functionality (Section 4.1). Moreover, we provide a first version of the algorithms that implement the functionality offered by the subsystems (Section 4.2).

4.1. Functionality of the Abstraction-oriented Service Base Management

The functionality offered by the AoSBM aims at sustaining the service discovery concern in the context of the FI of Services. In general, a typical service discovery process should provide at least the following core functionalities:

- `Publication of Service Offering`
- `Discovery of Service`

Having the typical discovery process as a basis, the AoSBM does provide the above mentioned functionalities whose initial perspective for the case of the FI was outlined in subsection 4.4 of the CHOReOS deliverable D1.2 [CHO11c]. In the current document, we revisit them in order to provide their thorough specification. Moreover, we describe functionalities that assist the `Discovery of Service` functionality to cope with the plenitude of business services providing efficient and effective searching facilities. As discussed in subsection 2.2, the plenitude of business services that is anticipated in the FI is a major scalability issue that the AoSBM should deal with. To this end, the AoSBM is based on organizing the available business service descriptions into automatically constructed groups. Specifically, the AoSBM organizes service descriptions in hierarchically structured groups of two types: functional and non-functional. A functional group consists of descriptions of services that offer the same/similar functionality and are described by a functional abstraction. In a similar vein, a non-functional group comprises services that are characterized by the same/similar non-functional properties and are represented by a non-functional abstraction.

To manipulate this large amount of service descriptions, the AoSBM takes into account that the available service descriptions may be stored in multiple sources that can/should be accesses through different Service Discovery Protocols (SDPs). Specifically, a source may be a registry/portal/directory that publishes information about business services offered by either a specific provider[1] or a massive number of providers[2]. Moreover, a source may be a set of devices (possibly even mobile) acting in a local network that offer business services. This situation introduces scalability, heterogeneity and mobility issues that should be handled by the AoSBM. Moreover, this situation implies an adaptability issue in the sense that the AoSBM should use sources desired by the AoSBM end-user. To deal with these issues, the AoSBM can be configured to use multiple Service Discovery Protocols (SDPs) that enable the retrieval of service descriptions from different sources, using the Plugin Manager of

[1] http://aws.amazon.com/
[2] http://webservices.seekda.com/

CHOReOS XSD described in the CHOReOS deliverable D3.1 [CHO11a]. For instance, the AoSBM may be configured to use a well-known Web services crawler and/or UPnP, SLP, etc. For further flexibility, the AoSBM is able to manage multiple sets of service descriptions that come from multiple sources selected according to the AoSBM end-users' preferences. Hereafter, we use the term *service collection* to refer to a particular set of service descriptions. Note that storing service descriptions multiple times can be easily avoided (e.g. by storing references to service descriptions).

The motivation behind our decision to manage multiple service collections from different sources is based on the fact that the AoSBM end-users that activate the discovery process may be interested in services offered by either specific (e.g. famous) providers or services located in a local network. As an example, we can indicatively report two cases of our running case study (subsection 2.2). In the first case, the AoSBM end-users are interested in services that serve for booking hotels that co-operate with their airplane company. These service descriptions may be found in specific service portals that are related to the AoSBM end-users' airplane company. In the second case, the AoSBM end-users are interested in services that serve for providing information about the local transportation system regarding buses that depart from the airport. The information about these services can be retrieved by searching in the local network of the airport. In this example, we observe that the AoSBM end-users are interested in services offered by different sources, namely, specific portals and a local network.

The purpose of Section 4.1 is to describe all the functionalities from an external point of view. In general, the external perspective of a software system is specified by the actors, which use the functionalities, and, the way in which the actors interact with the functionalities in terms of (a) the input information required for their initiation, (b) the data flow that follows their activation and (c) the outcome of their execution. Specifically, in each one of the following subsections 4.1.1 – 4.1.3, we detail a different functionality. Prior to turning to these subsections, we give the AoSBM architectural view, an overview of the functionalities offered by the AoSBM components and the kinds of the actors that initiate them.

AoSBM architecture and overview of the offered functionalities

Figure 4.1 gives the architecture of the AoSBM. The actors related with the AoSBM use its functionality by interacting with the `Service Registration` and the `Service Discovery` components. The `Service Registration` component controls the management activities over the contents of the `Abstraction Base`. These activities concern the creation/refreshing of a service collection and the (update of the) organization of a collection of services. To create/refresh a collection, the CHOReOS XSD is used in order to retrieve services from a target set of sources.

To organize the services of a collection, the `Abstraction-driven Service Organization` component is used. The latter component realizes a systematic approach that organizes a service collection into hierarchies of functional and non-functional abstractions. The produced hierarchies of abstractions are returned to the `Service Registration` component which stores them in the `Abstraction Base`.

The `Service Discovery` component is used whenever services are sought for participating in choreographies. The two modes of the discovery process are realized by the `Browsing Engine` and the `Querying Engine`. In both modes, the process is executed over the hierarchies stored in the `Abstraction Base`. To this end, these hierarchies along with the represented services are retrieved by interacting with the `Abstraction Base` component.

Specifically, the service collection management functionalities, which assist the `Discovery of Service` functionality, are the following:

- `Creating Service Collection` realized by the `Service Registration` component
- `Refreshing Service Collection` realized by the `Service Registration` component
- `Organization of Services` realized by the `Abstraction-driven Service`

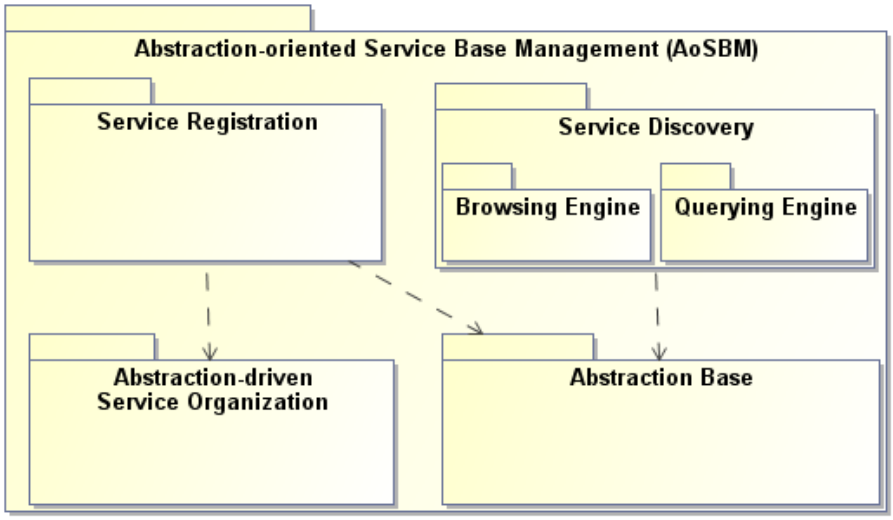


Figure 4.1: Architecture of the Abstraction-oriented Service Base Management.

`Organization` component

- `Updating the Organization of Services` realized by the `Abstraction-driven Service Organization` component.

The `Creation of Service Collection` functionality concerns the population of the AoSBM with service descriptions collected from new sources. As already being discussed, the identity of these sources is provided by the AoSBM end-users and following, the information is automatically obtained and stored in the AoSBM as a new service collection. Nevertheless, during the lifecycle of a source, the content of the source may change. In this case, a service collection that is bound to this source should update its contents in the AoSBM. The update of a service collection is realized by the `Refreshing Service Collection` functionality. The third functionality concerns the organization of the services of each collection into hierarchies of functional and non-functional abstractions by using the `Organization of Services` functionality. Moreover, in case the content of the collection has been refreshed, the hierarchies of abstractions, which organize the services of the collection, should also be updated by using the `Updating the Organization of Services` functionality. Finally, the `Discovery of Service` functionality is related to the provision of efficient and effective searching facilities over the hierarchies of functional and non-functional abstractions that organize the services of a collection.

Actors that interact with the offered functionalities

The actors that use the offered functionalities are of the following three types:

- `providers` of services
- `curator`[3] of the `Abstraction Base`
- `consumers` of services

In detail, the provider entity generally is a person or organization that offers some functionality through a particular service [W3C]. A provider can advertise its services through the AoSBM by interacting with the `Publication of Service Offering` activity. On the other hand, the curator conducts the four management activities (`Creating Service Collection`, `Refreshing Service`

[3] This is a typical term used in the Data Management community for the administrator.

Collection, Organization of Services, Updating the Organization of Services). Finally, the consumer entity activates the offered searching facilities over a service collection by interacting with the Discovery of Service functionality. In general, a consumer is a person that wishes to find out a service that meets requirements. Consumers can be domain experts, designers, developers or choreography users.

4.1.1. Creating/Refreshing Service Collection

We describe the two out of the four functionalities that realize management activities over the content of the AoSBM, while the remaining are detailed in the next subsection. All these functionalities are initiated by the curator. Specifically, we currently describe the Creating Service Collection and Refreshing Service Collection functionalities depicted in Figure 4.2 and 4.3 respectively.

Creating Service Collection

This functionality provides to the curator a list of the available sources (Form Target Set of Sources of the Collection activity). Following, the curator has the option to select from them none, one or more sources that will be included in the creation of the new collection (Select Sources from Existing Ones activity). Moreover, the curator may wish to include in the collection newly provided sources (Provide New Sources activity) whose services have not been previously retrieved. In this case, information about the sources is required by the curator, which concerns the *identities* of the sources (the *URIs* (Universal Resource Identifiers) of the sources) and/or the *multicast addresses*, and the *types* of the SDPs. The Retrieve Service Descriptions from Sources activity is used to massively retrieve service descriptions from the sources. Furthermore, among the services of a new or existing source, the curator has the option to include in the new service collection not all its services but a selected subset of them (Select Service Descriptions activity). In total, the information about the new service collection is stored in the AoSBM. All the actions performed for storing the new collection in the AoSBM are included in the Store Service Collection activity.

Refreshing Service Collection

Turning to the second functionality, in a similar vein as the previous one, it requires as input by the curator the target service collection among the already defined ones. In particular, the functionality provides to the curator a list of the defined service collections, which are uniquely identified by their constituent sources and, following, the curator selects one of these collections (Select Target Service Collection activity). The purpose of this functionality is to refresh the contents of the collection. To this end, the Configure Suitable SDPs and Retrieve Service Descriptions from Sources activity is used in order to massively retrieve the current content of the sources of the collection.

The newly retrieved content may differ from the previous one and the service collection is accordingly modified. The information about the new service collection is stored in the AoSBM (Store Refreshed Service Collection activity). Finally, we note that the existing hierarchies of abstractions that organize services of a refreshed collection are updated on demand by the curator by using the Updating the Organization of Services functionality (described in the following subsection).

4.1.2. Organization of Services

As a continuity from the previous subsection, we turn to the next functionalities that realize the two remaining management activities over the content of the AoSBM: the Organization of Services and the Updating the Organization of Services, as depicted by Figure 4.4 and 4.5 respectively.

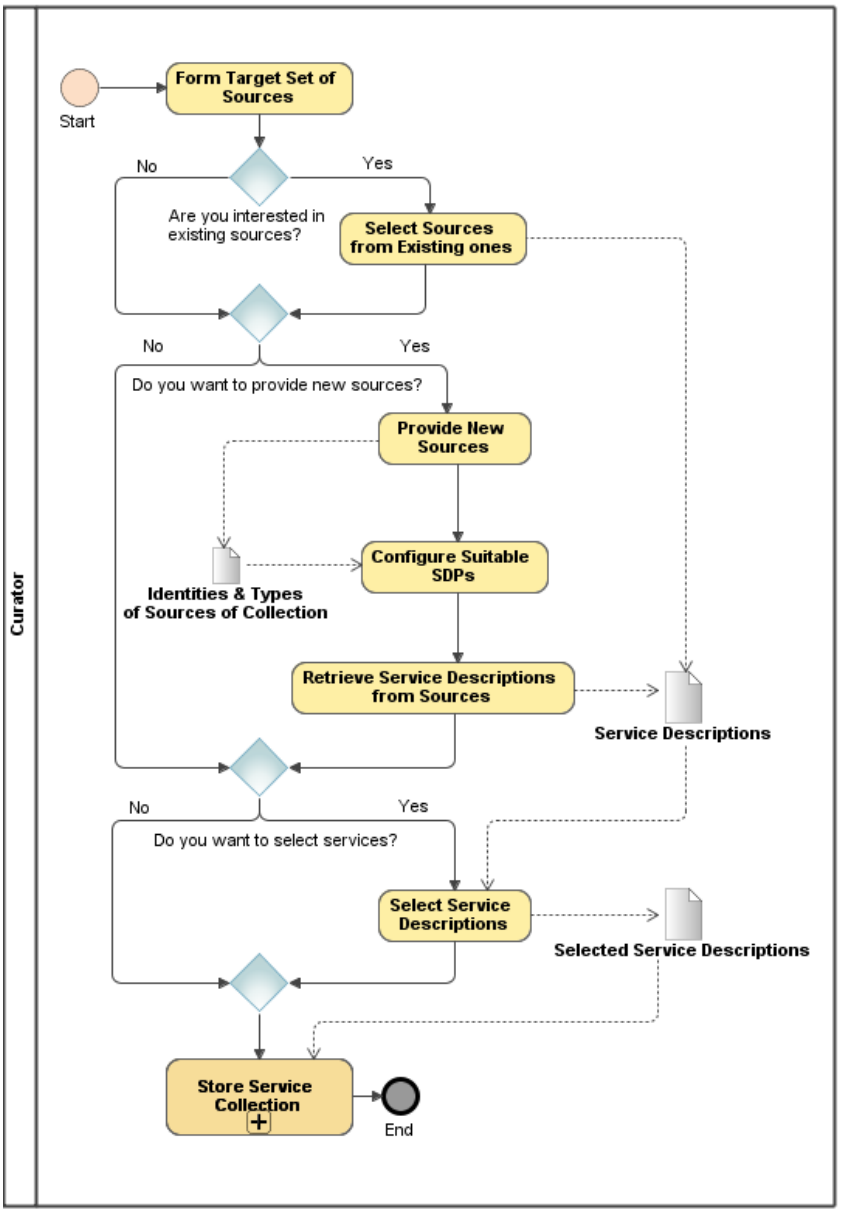


Figure 4.2: Process model of the `Creating Service Collection` functionality.

Organization of Services

The first functionality accepts as input by the curator a collection of services that will be organized into hierarchically structured groups. Specifically, the functionality provides to the curator a list of the existing collections and the curator selects one of them (`Select Target Service Collection` activity). Each produced group is described by a representative called abstraction. To organize the services of the selected collection into groups of similar services and produce abstractions, the functionality uses a systematic approach. This approach compares the service descriptions in order to find similar ones. Given that a service description consists of two parts, the functional and the non-functional, services can offer similar functionality and similar non-functional properties. Therefore, the approach uses two different kinds of metrics for evaluating the similarity degree among the services in terms of the functional and non-functional part of their description. Consequently, using the two similarity metrics, the approach produces two different types of organizations of services. The first type is a hierarchy of functional abstractions whereas, the second type is a hierarchy of non-functional abstractions.

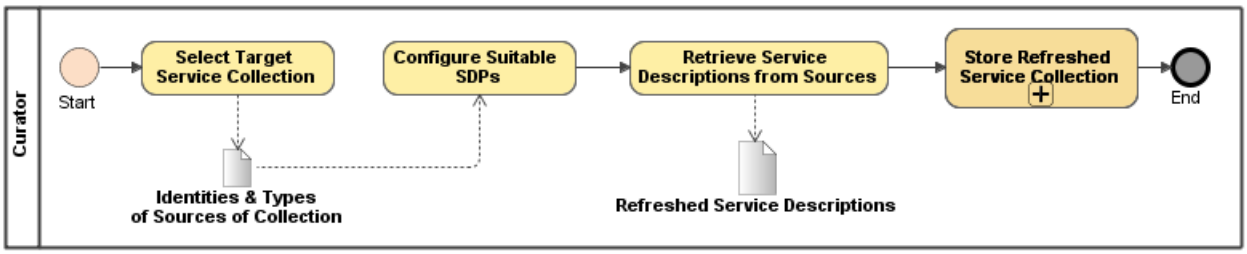


Figure 4.3: Process model of the `Refreshing Service Collection` functionality.

However, taking into consideration that the two kinds of organizations classify the same set of services, the emerging issue is whether these two organizations are jointly or independently developed. Overall, three options/*modes* can be realized as follows.

a) The functional and the non-functional abstractions are *jointly* developed so that

   i) services represented by a functional abstraction are further organized with respect to nested non-functional abstractions

   ii) services represented by a non-functional abstraction are further organized with respect to nested functional abstractions.

b) The functional and non-functional abstractions are *independently* developed.

We anticipate that the first option provides organizations that would be useful in typical service discovery scenarios where the similarity of the non-functional properties of services is meaningful only if the services are functionally similar. On the other hand, the other two options may be interesting in special cases (e.g. the AoSBM end-user looking for statistics concerning the availability of services that offer different functionalities). The curator may choose any of the aforementioned modes (`Select Organization Mode` activity). All the actions performed for organizing services based on different kinds of hierarchies are included in the `Produce Hierarchies of Abstractions` activity.

To conclude, the information about the produced groups of services, the relations among groups and the abstractions that represent the groups is finally stored in the `Abstraction Base`. All these actions performed for storing the hierarchies of abstractions into the `Abstraction Base` are included in the `Store Hierarchies into the Abstraction Base` activity.

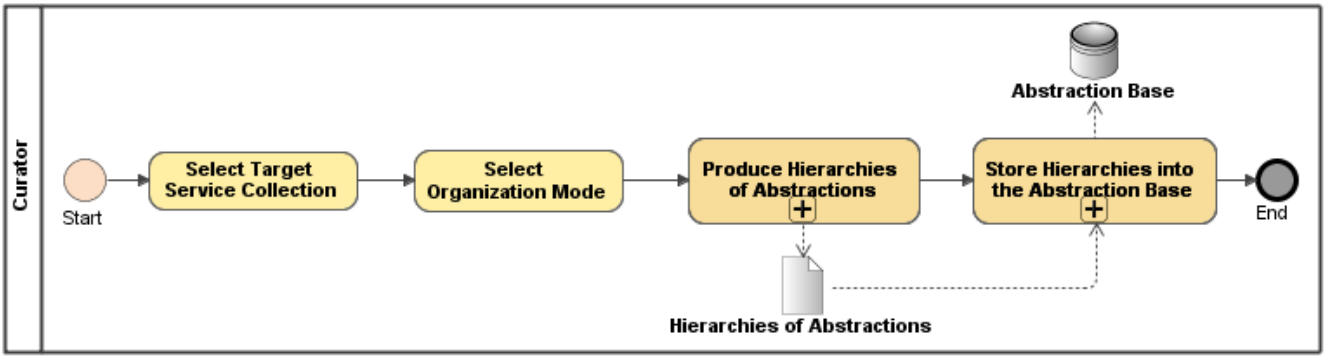


Figure 4.4: Process model of the `Organization of Services` functionality.

Updating the Organization of Services

The curator may update the hierarchies of abstractions that organize a collection of services. The need for this functionality is motivated by the fact that the content of a collection may change and, consequently, the existing abstractions may not reflect the new content of the collection.

The functionality provides to the curator a list of the available collections and the curator selects one of them (`Select Target Service Collection` activity). To update the existing hierarchies that organize the services of the selected collection, the functionality needs the new version of the content of the collection. This new version have already been retrieved by activating the `Refreshing Service Collection` functionality detailed in the previous subsection. To perform the update, the functionality firstly compares the new version of the collection to the previous one. The two versions may differ in three ways which are described in the following part along with the actions performed in each case in order to update the existing hierarchies (`Update Hierarchies Incrementally` activity). The information about no longer available services are removed from the functional and the non-functional hierarchies whereas, information about new service descriptions is added in functional and non-functional abstractions of the hierarchies. The registration of new information in the hierarchies may fail if the hierarchies do not contain suitable abstractions. Dealing with such situations amounts to re-building the hierarchies (`Re-build Hierarchies` activity). To conclude, the updated versions of the hierarchies (`Updated Hierarchies` object) are stored in the `Abstraction Base` by using the `Store Updated Hierarchies into the Abstraction Base` activity.

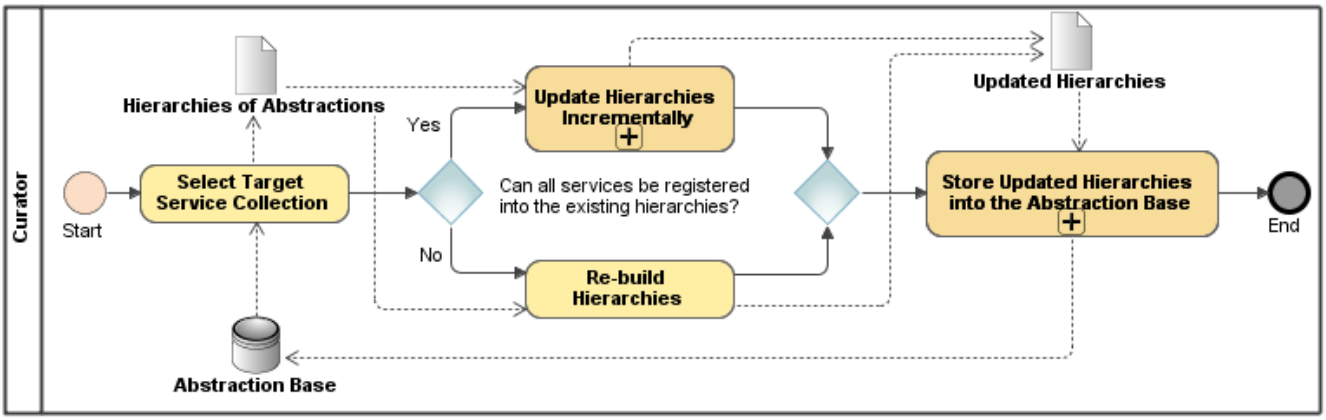


Figure 4.5: Process model of the `Updating the Organization of Services` functionality.

4.1.3. Discovery of Service

The `Discovery of Service` functionality (depicted in Figure 4.6) is initiated by consumers (domain experts, designers and developers) who are in need of discovering appropriate services for choreographies. Given that the choreographies are designed and developed based on service abstractions, this functionality returns the abstractions and the represented concrete services that meet the requirements (service demand) for the choreographies.

The consumers can select the mode in which the discovery process will be performed. The potential modes are *searching* and *browsing* realized by the `Search for Services` and the `Browse for Services` activity respectively. The details of these activities will be described in future deliverables in which the searching and the browsing process over the `Abstraction Base` is presented. In this point, we give an overview of the process followed in these activities.

Specifically, in the searching mode, the consumers specify their requirements about the desired services (`Provide Searching Criteria` activity). These criteria are aligned in the form of a structured query which is formally defined in the CHOReOS deliverable D1.3 [CHO11d]. Such a query mainly

specifies restrictions over the functional and the non-functional properties of the desired services. The query is executed over the existing hierarchies of functional and non-functional abstractions that organize the services of the selected collection and have been retrieved from the `Abstraction Base`. The output of the query is a set of alternative concrete services that satisfy the restrictions specified by the query. In the output, the concrete services are accompanied by the functional and/or the non-functional abstractions that describe these services (`Abstractions&Represented Services` object). These abstractions may be interrelated or not, depending on the mode based on which the services of the collection were initially organized. We note that the output of a query may include a set of many alternative concrete services whereas, at runtime, consumers (in particular choreography users) may are in need of one of them. In this case, the consumers have the option to switch to the browsing mode and navigate in the abstractions in order to select one of the represented concrete services.

In the browsing mode, the hierarchies of abstractions that organize the services of the selected collection are presented to the consumers. Based on their requirements, the consumers traverse these hierarchies in order to compare the properties of the abstractions with their requirements. The traversal is realized by expanding or collapsing the abstractions of the hierarchies. The expansion of an abstraction reveals represented concrete services or further lower-level abstractions. Moreover, if the functional and non-functional hierarchies have been jointly developed, then the expansion of a functional abstraction (resp. non-functional) reveals nested non-functional (resp. functional) abstractions. Consumers may undo expanding operations by collapsing abstractions. All these actions performed in the browsing process are included in the `Browse for Services` activity.

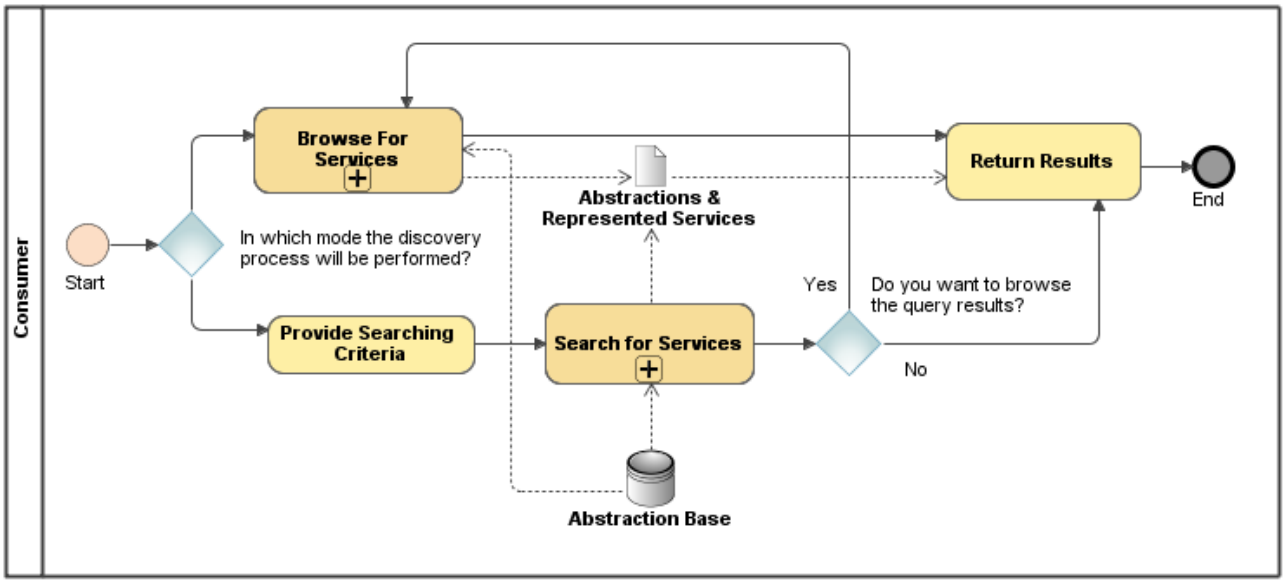


Figure 4.6: Process model of the `Discovery Of Service` functionality.

4.2. Abstraction-oriented Service Organization Algorithms

In this section, we discuss an initial suite of algorithms that implement the core functionalities for the organization of business services. The services are organized in hierarchies of abstractions that are of two different types, functional and non-functional; we discuss corresponding algorithms in subsections 4.2.1 and 4.2.2.

4.2.1. Organizing Services into Hierarchies of Functional Abstractions

In this subsection, we discuss an algorithm that organizes services in hierarchies of functional abstractions. In the following subsections, we briefly recall the general definition and notations used for modeling the concept of services and the concept of functional abstractions that are defined in detail in D1.3 [CHO11d]. Following, we detail the *modus operandi* of the initial version of the algorithm.

Basic concepts

The organization of services with respect to functional abstractions is based on clustering service interfaces. Following, we recall from the CHOReOS deliverable D1.3 [CHO11d] the definition of the concept of a service interface, along with the notation used to refer to it.

A service interface, i, is formally defined as a tuple (Definition 4.1), which consists of the following elements:

- a name ($i.n$) that characterizes the interface
- an optional profile ($i.p$) that corresponds to a user-intuitive explanation of the interface
- a set of operations ($i.O$ - Definition 4.2) that correspond to different functionalities provided through the interface
- an optional behavioral specification ($i.b$) that abstracts the usage of the functionalities of the interface
- an optional set of constraints ($i.C$ - Definition 4.7) related to requirements over the environment where the interface functionalities execute.

An operation, op, of an interface is also defined as a tuple (Definition 4.3) that comprises:

- a name ($op.n$) for the operation
- an optional profile ($op.p$) that contains a user-intuitive explanation of the operation
- a set of input parameters ($op.In$ - Definition 4.4) and a set of output parameters ($op.Out$ - Definition 4.5)
- an optional pre-condition ($op.pre$) that must hold for the correct execution of the operation
- an optional post-condition ($op.post$) that shall hold after the correct execution of the operation.

Finally, a parameter, par, of an operation is defined as a tuple (Definition 4.6), which comprises the following elements:

- a name ($par.n$) for the parameter
- an optional profile ($par.p$) that contains a user-intuitive explanation of the parameter
- a XML data type ($par.t$) for the parameter.

According to the previous definition, certain elements of a service interface are optional (e.g., profiles, pre/post conditions). Since these elements are rarely provided in practice, the proposed algorithm that organizes services with respect to functional abstractions takes into account the compulsory elements of service interfaces. Taking into account the optional elements would unnecessarily increase the complexity of the algorithm and this issue is of particular importance considering the large amount of services that we anticipate in the FI.

$$[Interface]\ \ i = (n,\ p,\ O,\ b,\ C) \tag{4.1}$$
$$[Operations]\ \ O = \{op_1,\ \ldots,\ op_{|O|}\} \tag{4.2}$$
$$[Operation]\ \ op = (n,\ p,\ In,\ Out,\ pre,\ post)\ |\ op \in O \tag{4.3}$$
$$[Input\ message]\ \ In = \{par_1,\ \ldots,\ par_{|In|}\} \tag{4.4}$$
$$[Output\ message]\ \ Out = \{par_1,\ \ldots,\ par_{|Out|}\} \tag{4.5}$$
$$[Parameter]\ \ par = (n,\ p,\ t)\ |\ par \in In\ \vee\ par \in Out \tag{4.6}$$
$$[Constraints]\ \ C = \{c_1,\ \ldots,\ c_{|C|}\} \tag{4.7}$$

Figure 4.7: Definition of the interface of a service.

As already discussed, a functional abstraction represents the common/similar functionalities, realized by operations, provided by a set of services. In [CHO11d] the formal definition of this concept is based on the fundamental notion of behavioral sub-typing introduced by Liskov & Wing defined in [LW94]. According to Liskov & Wing, two types have a correct sub-typing relation if and only if a number of fundamental rules are guaranteed. In the CHOReOS deliverable D1.3 [CHO11d] we adapted these rules in our context and discussed in detail the definition of functional abstractions. Here, we briefly underline the key factors of the definition of functional abstractions that are necessary in order to describe in the remainder the algorithm that automatically extracts such abstractions.

Roughly speaking, the key factors of a functional abstraction are: (a) the services that the abstraction represents, (b) the interface that represents the common/similar functionalities offered by the represented services, (c) the way that the interface of the abstraction is mapped to the interfaces of the services, (d) other lower and higher level functional abstractions associated with the current abstraction given that the produced abstractions formed a hierarchy.

More concretely, a functional abstraction, fa, is a tuple (Definition 4.8) that comprises:

- a set of services ($fa.R$ - Definition 4.9), which are represented by the abstraction
- an interface ($fa.i$) whose operations correspond to common/similar operations offered by the interfaces of the represented services
- a set of interface mappings ($fa.M$ - Definition 4.10) between the interface of the abstraction, $fa.i$, and the interfaces, ri_j, of the represented services, $s_j \in fa.R$
- a set of lower level functional abstractions ($fa.desc$) that represent subsets of the represented services
- a higher level functional abstraction ($fa.anc$) the represent a superset of the represented services.

Based on the previous definition, the proposed algorithm constructs clusters of services that provide common/similar functionalities and for each cluster it defines a corresponding functional abstraction. Concerning the mappings between the interface of the functional abstraction and the interfaces of the represented services adapt/precise the general definition of the CHOReOS deliverable D1.3 [CHO11d] as follows.

A mapping $m_{ri_i} = (m_{op}, M_{In}, M_{Out}) \in fa.M$ between $fa.i$ and the interface ri_i of a represented service s_i is defined as a tuple that consists of:

- A one-to-one function $m_{ri_i}.m_{op} : fa.i.O \rightarrow ri_i.O$ between the operations $fa.i.O$ of the abstract interface and the operations $ri_i.O$ of the represented interface.

- A set of mappings $m_{ri_i}.M_{In}$ between the input parameters of mapped operations and a set of mappings $m_{ri_i}.M_{Out}$ between the output parameters of mapped operations. In particular, given the mapping $m_{ri_i}.m_{op} : fa.i.O \rightarrow ri_i.O$ between the operations $fa.i.O$ of the abstract interface and the operations $ri_i.O$ of the represented interface ri_i, for each pair of mapped operations $op \in fa.i.O$ and $m_{ri_i}.m_{op}(op) \in ri_i.O$ we have:
  - $m_{ri_i}.M_{In}$ contains a one-to-one function $m_{in} : op.In \rightarrow m_{ri_i}.m_{op}(op).In$ for the inputs of the mapped operations.
    With respect to the Liskov & Wing contra-variance rule, we assume that m_{in} is well-formed if for every pair of mapped input parameters $p \in op.In$ and $m_{in}(p)$ the type of p is a sub-type of the type of $m_{in}(p)$.
  - $m_{ri_i}.M_{Out}$ contains a one-to-one function $m_{out} : op.Out \rightarrow m_{ri_i}.m_{op}(op).Out$ for the outputs of the mapped operations.
    With respect to the Liskov & Wing co-variance rule, we assume that m_{out} is well-formed if for every pair of mapped output parameters $p \in op.Out$ and $m_{out}(p)$ the type of $m_{out}(p)$ is a subtype of the type of p.

Note that the sub-typing relation for parameters is defined with respect to the standard XML type hierarchy (Figure 4.9). Based on this hierarchy, we use the relation, $t_1 \subseteq t_2$, to denote that either t_1 is equivalent with t_2 (e.g., both types correspond to the XML `string` build-in type), or t_1 is a subtype of t_2 (e.g., t_1= `normalizedString` which is subtype of t_2 = `string`).

$$[Functional\ abstraction]\ \ fa = (i, R, M, anc, desc) \tag{4.8}$$
$$[Represented\ services]\ \ R = \{si_1,\ \ldots,\ si_{|R|}\} \tag{4.9}$$
$$[Interface\ mappings]\ \ M = \{m_{ri_1},\ \ldots,\ m_{ri_{|M|}} \mid ri_j = si_j.i\ \wedge\ si_j \in R\} \tag{4.10}$$
$$[Interface\ mapping]\ \ m_{ri_j} = (m_{op},\ M_{In},\ M_{Out}) \mid m_{ri_j} \in M \tag{4.11}$$
$$[Operations\ mapping]\ \ m_{op} : fa.i.O \rightarrow ri_j.O \tag{4.12}$$
$$[Input\ mapping]\ \ m_{in} : op.In \rightarrow m_{op}(op).In \mid op \in fa.i.O\ \wedge\ m_{in} \in M_{In} \tag{4.13}$$
$$[Output\ mapping]\ \ m_{out} : op.Out \rightarrow m_{op}(op).Out \mid op \in fa.i.O\ \wedge\ m_{out} \in M_{Out} \tag{4.14}$$

Figure 4.8: Definition of the notion of functional abstraction.

Producing hierarchies of functional abstractions

The proposed algorithm accepts as input a set of services and its unitary step is the definition of functional abstractions for these services. The given set of services maybe a service collection (Section 4.1) or a subset of such a collection (e.g., a set of services that is represented by a non-functional abstraction). Each functional abstraction represents a group of services that provide a common/similar set of functionalities that are offered by the represented services. In general, identifying services that provide semantically similar functionalities is hard. However, as discussed in [AZVI11], we empirically observed that *it is very frequently encountered to have semantically similar services that provide syntactically similar interfaces*. Our preliminary experiments described in [AZVI11] provide a practical evidence for this correlation. Moreover, in [AZVI11] we observed that the identification of semantically similar services also depends on the characteristics of the given set of services, e.g., the false positives increase as the size and diversity of a given set of services increase.

Overall, the proposed algorithm divides the given set of services into smaller groups of similar services. Following, the algorithm mines abstractions, which represent the interfaces of the services of

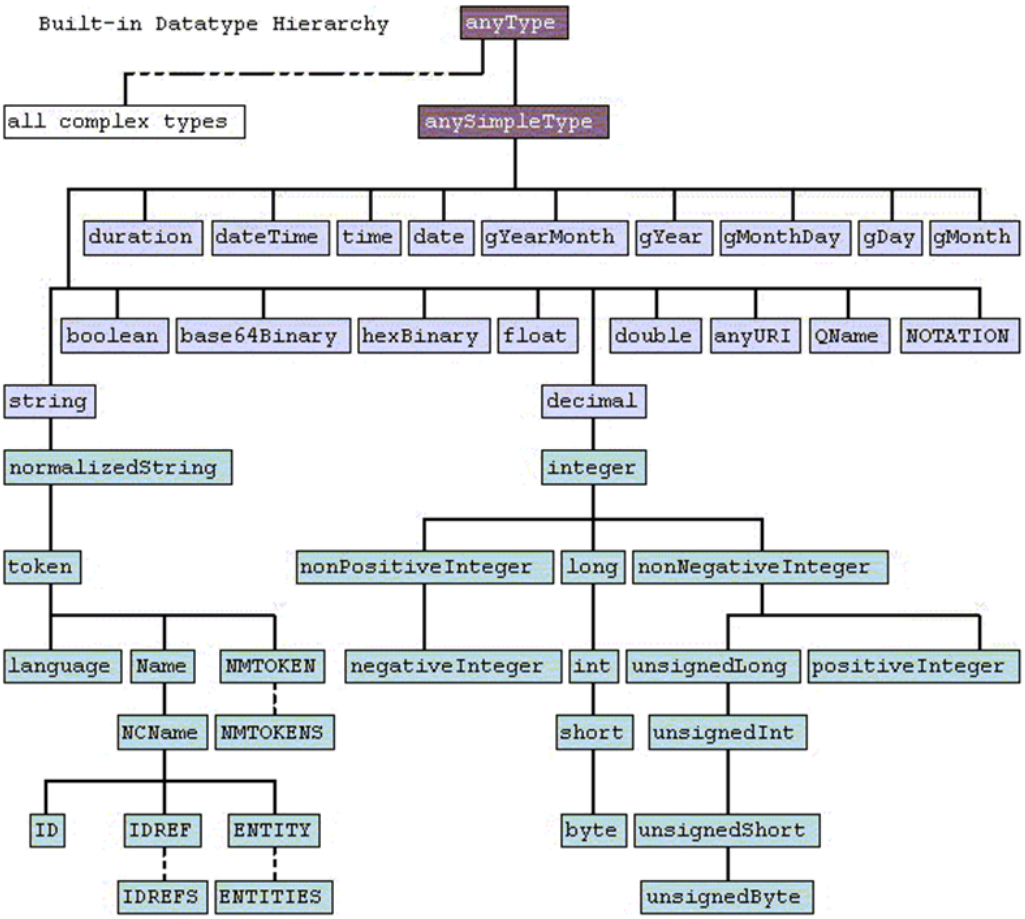


Figure 4.9: The standard XML type hierarchy [W3C04].

each group and organizes these abstractions in hierarchies. To this end, the following two main issues emerge: (1) the definition of a metric that quantifies the similarity between service interfaces; (2) the construction of the hierarchy of functional abstractions.

Distance between service interfaces: We compute the distance between two interfaces, i_1 and i_2, based on the compulsory elements of these interfaces via a metric defined in terms of the following elements.

- The names of the two interfaces ($i_1.n$ and $i_2.n$).
- The operations of the two interfaces. For each operation ($op \in i_1.O$ or $op \in i_2.O$), we take into account:
  - The name of the operation ($op.n$).
  - The input and output parameters ($op.In$ and $op.Out$) of the operation. For each parameter ($par \in In$ or $par \in Out$), we consider the name ($par.n$) and the type ($par.t$) of the parameter.

More formally, the proposed metric, D_i, is defined as follows (Definition 4.15): given the two interfaces, i_1 and i_2, and a mapping between their operations, the distance, $D_i(i_1, i_2)$, is the arithmetic mean of the following two terms:

a) The normalized edit distance between the names of the interfaces ($NED(i_1.n, i_2.n)$)[4].

[4] A simple way to calculate the distance between names is to consider them as simple strings and to calculate their

b) The arithmetic mean of the distances between the mapped operations ($\overline{D_{op}}$).

The operations distance, D_{op}, is defined as follows (Definition 4.17). Given two operations, op_1 and op_2, the distance, $D_{op}(op_1, op_2)$, is the arithmetic mean of the following two terms:

a) The normalized edit distance between the names of the operations calculated by using the length of their *longest common substring* described earlier ($NED(op_1.n, op_2.n)$).

b) The arithmetic mean of the distances between the input and output messages of the operations ($D_{io}(op_1, op_2)$ - Definition 4.18).

Similarly, given two messages, m_1 and m_2, and a mapping between the parameters of the messages, the distance, $D_m(m_1, m_2)$, is defined as the arithmetic mean of the following two terms (Definition 4.19):

a) The normalized edit distance between the names of the messages calculated by using the length of their *longest common substring* described earlier ($NED(m_1.n, m_2.n)$).

b) The arithmetic mean of the distances between the mapped parameters of the messages ($\overline{D_p}$).

The distance $D_p(p_1, p_2)$, between two parameters p_1 and p_2 is defined as the arithmetic mean of the two following terms (Definition 4.21):

a) The normalized edit distance between the names of the parameters calculated by using the length of their *longest common substring* described earlier ($NED(p_1.n, p_2.n)$).

b) The distance between the built-in types of the parameters ($D_t(p_1.t, p_2.t)$). Specifically, in case the types $p_1.t, p_2.t$ lie on the same branch of the hierarchy, $D_t(p_1.t, p_2.t)$ is the *absolute subtraction* of the depths of these types in the standard XML type hierarchy (Figure 4.9) normalized by the maximum height ($maxHeight$) of the hierarchy; otherwise, we assume that the types are incompatible and the distance $D_t(p_1.t, p_2.t)$ is *infinity*.

In practice, to calculate the value of the distance between two interfaces, we have to find the best possible operations mapping, namely, the mapping between the most similar operations of the interfaces. To this end, the most similar pairs of operations are found by reducing this problem in a well-known optimization problem: *the maximum weighted matching problem in a bipartite graph* [Mun57]. The nodes of the graph correspond to the operations of the interfaces, while the edges correspond to the distances of pairs of operations. Similarly, to calculate the distance between two messages we need to determine the best possible message mapping, namely, the mapping between the most similar parameters of the messages. The most similar pairs of parameters are found by solving again the maximum weighted matching problem in the bipartite graph, in which the nodes correspond to the parameters of the messages while, the edges correspond to the distances between pairs of parameters of the messages.

To illustrate the calculation of the interface distance, we take an example in which we compute the value of the distance between the interfaces, `sendSmsHttpPost` and `SMSTextMessagingSoap`, offered by two real-world services, `SendSms`[5] and `SMSTextMessaging`[6] respectively. These interfaces provide the following operations:

Interface : `sendSmsHttpPost`

(normalized) edit distance. In detail, the edit distance between two strings, s_1 and s_2, with lengths, n and m respectively, can be defined as: $ED(s_1, s_2) = n + m - 2 * lcs(s_1, s_2)$, where $lcs(s_1, s_2)$ is the length of their *longest common substring*. Their normalized edit distance equals to $NED(s_1, s_2) = \frac{2*ED(s_1,s_2)}{n+m+ED(s_1,s_2)}$. Note that we intend to investigate alternative ways for calculating distance between names in future versions of the algorithm.

[5] http://www.sms-txt.co.uk/sendSms.asmx

[6] http://ws.strikeiron.com/StrikeIron/globalsmspro2_5/SMSTextMessaging

$$D_i(i_1,\ i_2) = \frac{NED(i_1.n,\ i_2.n) + \overline{D_{op}}}{2} = \frac{NED(i_1.n,\ i_2.n) + \frac{\sum_{\forall(op_i,\ op_j)\in M_{op}} D_{op}(op_i,\ op_j)}{|M_{op}|}}{2} \quad (4.15)$$

$$M_{op} = \{(op_i,\ op_j) \in i_1.O \times i_2.O\} : \quad (4.16)$$
$$\{(op_i,\ op_j)\} \subset i_1.O \times i_2.O\ \wedge \sum_{\forall(op_i,\ op_j)} D_{op}(op_i,\ op_j)\ is\ minimized$$

$$D_{op}(op_1,\ op_2) = \frac{NED(op_1.n,\ op_2.n) + D_{io}(op_1,\ op_2)}{2} \quad (4.17)$$

$$D_{io}(op_1,\ op_2) = \frac{D_m(op_1.In,\ op_2.In) + D_m(op_1.Out,\ op_2.Out)}{2} \quad (4.18)$$

$$D_m(m_1, m_2) = \frac{NED(m_1.n, m_2.n) + \overline{D_p}}{2} = \frac{NED(m_1.n, m_2.n) + \frac{\sum_{\forall(p_i,p_j)\in M_m} D_p(p_i,p_j)}{|M_m|}}{2} \quad (4.19)$$

$$M_m = \{(p_i,\ p_j) \in m_1.O \times m_2.O\} : \quad (4.20)$$
$$\{(p_i,\ p_j)\} \subset m_1.O \times m_2.O\ \wedge \sum_{\forall(p_i,\ p_j)} D_p(p_i,\ p_j)\ is\ minimized$$

$$D_p(p_1,\ p_2) = \frac{NED(p_1.n,\ p_2.n) + D_t(p_1.t,\ p_2.t)}{2} \quad (4.21)$$

$$D_t(p_1.t, p_2.t) = \begin{cases} \frac{|p_1.t - p_2.t|}{maxHeight}, if\ the\ types\ belong\ to\ the\ same\ branch\ of\ the\ hierarchy \\ \infty, otherwise \end{cases} \quad (4.22)$$

Figure 4.10: Distance metric for a pair of interfaces.

Method : `SendSms()`

Interface : `SMSTextMessagingSoap`

Method 1 : `SendMessage()`
Method 2 : `SendMessagesBulk()`
Method 3 : `TrackMessage()`
Method 4 : `TrackMessagesBulk()`
Method 5 : `GetSupportedCarriers()`
Method 6 : `GetCountryCodes()`

The value of the distance between these interfaces is, $D_i = 0.76$, which is the arithmetic mean of the distance of their names ($NED = 0.91$) and the distance between their mapped operations ($D_{op} = 0.60$). The best possible mapping between the operations relates the `SendSms()` operation of the first interface with the `SendMessage()` operation from the second interface. To determine this pair of mapped operations, the algorithm firstly defines based on all the operations of the two interfaces the bipartite graph depicted in Figure 4.11 and following, solves the maximum weighted matching problem in this graph. All the pair-wise operation distances are given in Table 4.1.

The value of the distance between the `SendSms()` and `SendMessage()` operations equals to $D_{op} = 0.60$, which is the arithmetic mean of the distance of their names ($NED = 0.71$) and the arithmetic mean of the distances between their input and output messages, $D_{io} = 0.50$ (Figure 4.12).

The value of the distance between the input messages, `SendSmsSoapIn` and `SendMessageSoapIn`, is equal to $D_m(In) = 0.36$, which is the arithmetic mean of (a) the

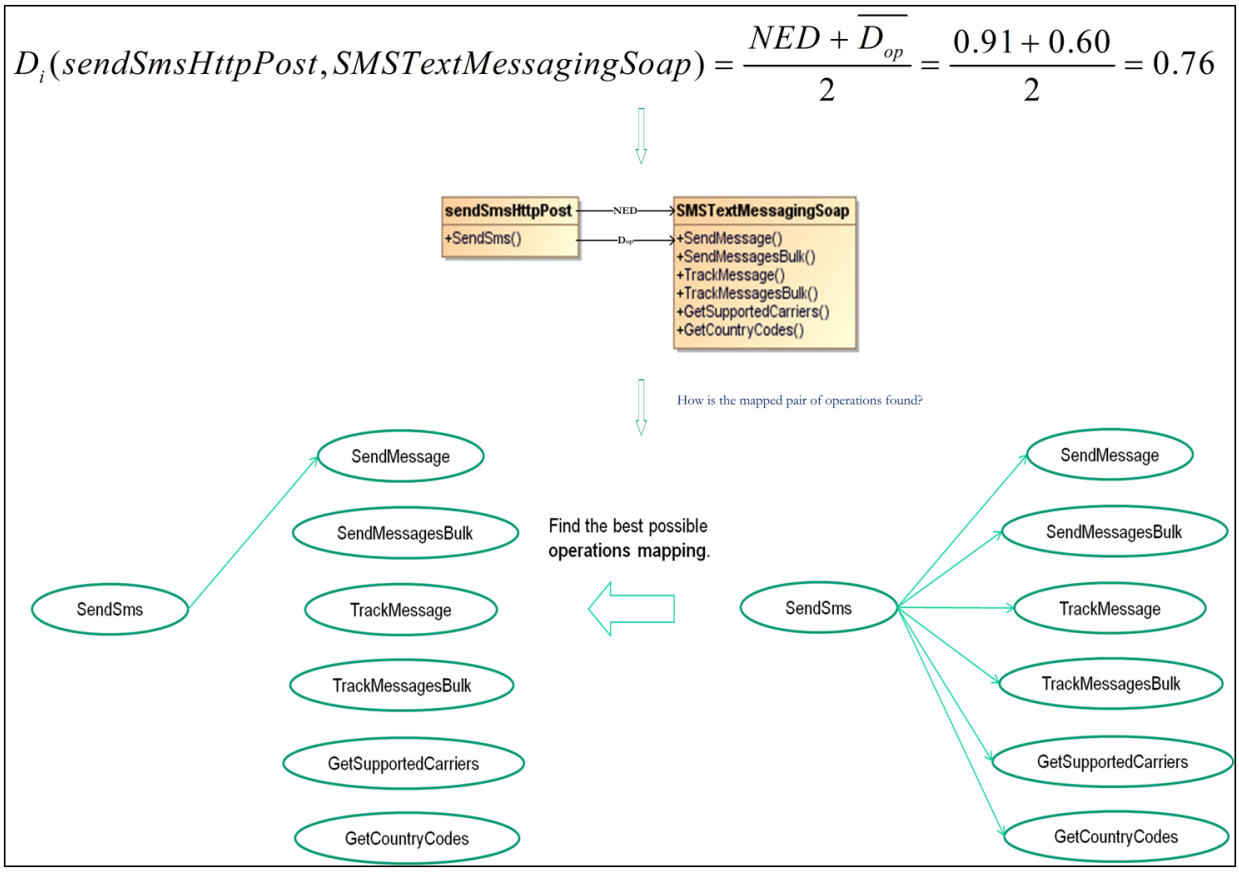


Figure 4.11: Example in calculating the distance between two interfaces.

distance of their names ($NED = 0.63$) and (b) the arithmetic mean of the distances of the pairs of their mapped parameters ($\overline{D_p} = 0.09$). To identify the pairs of the mapped parameters, the algorithm firstly defines the bipartite graph depicted in Figure 4.13 based on the parameters of the two input messages and following, solves the maximum weighted matching problem in this graph. All the pair-wise parameter distances are given in Table 4.2.

The value of the distance between the output messages, `SendSmsSoapOut` and `SendMessageSoapOut`, equals to $D_m(Out) = 0.63$, in which the distance of their names ($NED = 0.63$) and the mean of the distances of the mapped parameters ($\overline{D_p} = 0.00$). The distance between the parameters is zero because the output message of the operation `SendSms` is empty and hence, the mapping between the output messages of the two operations is trivial.

Construction of the hierarchy: Given that our goal is to identify groups of services that provide similar functionalities and define abstractions for these groups, our algorithm is based on clustering. Moreover, since we aim at a hierarchical organization of services we use agglomerative hierarchical clustering, which starts from *singleton* clusters (consisting of exactly one entity) and progressively, merges clusters together until only one cluster remains [MB07]. To adapt this general process in the case of the service interfaces, the proposed algorithm starts from the interfaces (considered as singleton clusters) and successively, merges (singleton or non-singleton) clusters of interfaces.

In particular, in our algorithm, we adopt the aforementioned general-purpose steps of the typical clustering process along with the distance metric between interfaces that we previously defined. We also extend these steps in order to not only form clusters of service interfaces but also extract a functional abstraction that represents each formed cluster. Moreover, given that the algorithm produces a hierarchy of groups of interfaces, it further produces a hierarchy of functional abstractions that represent these groups. Our extended clustering process is outlined in Algorithm 1 and its steps are detailed as

Table 4.1: Distance values between the operations of the interfaces `sendSmsHttpPost` and `SMSTextMessagingSoap`.

		sendSmsHttpPost
		SendSms()
SMSTextMessagingSoap	SendMessage()	(0.71 + 0.60) / 2 = **0.66**
	SendMessagesBulk()	(0.79 + 0.61) / 2 = 0.70
	TrackMessage()	(0.94 + 0.60) / 2 = 0.77
	TrackMessagesBulk()	(0.96 + 0.62) / 2 = 0.79
	GetSupportedCarriers()	(0.96 + 0.39) / 2 = 0.68
	GetCountryCodes()	(0.95 + 0.39) / 2 = 0.67

$$D_{op}(SendSms, SendMessage) = \frac{NED + D_{io}}{2} = \frac{NED + \frac{D_m(In) + D_m(Out)}{2}}{2} = \frac{0.71 + \frac{0.36 + 0.63}{2}}{2} = 0.60$$

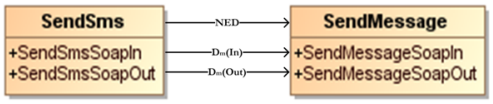


Figure 4.12: Example in calculating the distance between two operations.

follows.

The algorithm maintains a list of functional abstractions that represent the clusters formed at each step, $L = \{fa_1, \ldots, fa_{|L|} \mid fa_j : FunctionalAbstraction\}$. The clustering process is divided in two distinct phases. The first phase is for initialization and comprises the following two simple steps. In the first step (**Step 1** of Algorithm 1), singleton clusters are built for the the given service interfaces. For each singleton cluster, a *dummy* functional abstraction, is defined. We use the term *dummy* in order to emphasize on the fact that each one of these abstractions does not represent a set of interfaces but only one interface. Consequently, during the definition of a dummy abstraction, fa_j, the interface of the abstraction ($fa_j.i$) is the corresponding input interface, i_j. The dummy abstractions are inserted in the list, L, initializing its contents. In the second step, the distance between every pair of service interfaces is calculated initializing the contents of a two-dimensional distance matrix, d (**Step 2** of Algorithm 1).

Continuing with the second phase, the algorithm iteratively performs the following steps. In the first step, a candidate pair of functional abstractions pair of interfaces of functional abstractions is selected

Table 4.2: Distance values between the parameters of the input messages `SendSmsSoapIn` and `SendMessageSoapIn`.

		SendMessageSoapIn			
		FromName:String	FromNumber:String	ToNumber:String	MessageText:String
SendSmsSoapIn	FromName:String	(0.00 + 0.00) / 2 = **0.00**	(0.62 + 0.00) / 2 = 0.62	(0.93 + 0.00) / 2 = 0.93	(0.94 + 0.00) / 2 = 0.94
	FromNumber:String	(0.62 + 0.00) / 2 = 0.62	(0.00 + 0.00) / 2 = **0.00**	(0.50 + 0.00) / 2 = 0.50	(0.95 + 0.00) / 2 = 0.95
	ToNumber:String	(0.93 + 0.00) / 2 = 0.93	(0.50 + 0.00) / 2 = 0.50	(0.00 + 0.00) / 2 = **0.00**	(0.94 + 0.00) / 2 = 0.94
	Message:String	(0.93 + 0.00) / 2 = 0.93	(0.94 + 0.00) / 2 = 0.94	(0.93 + 0.00) / 2 = 0.93	(0.36 + 0.00) / 2 = **0.36**
	locale:String	(0.93 + 0.00) / 2 = 0.93	(0.93 + 0.00) / 2 = 0.93	(0.92 + 0.00) / 2 = 0.92	(0.93 + 0.00) / 2 = 0.93

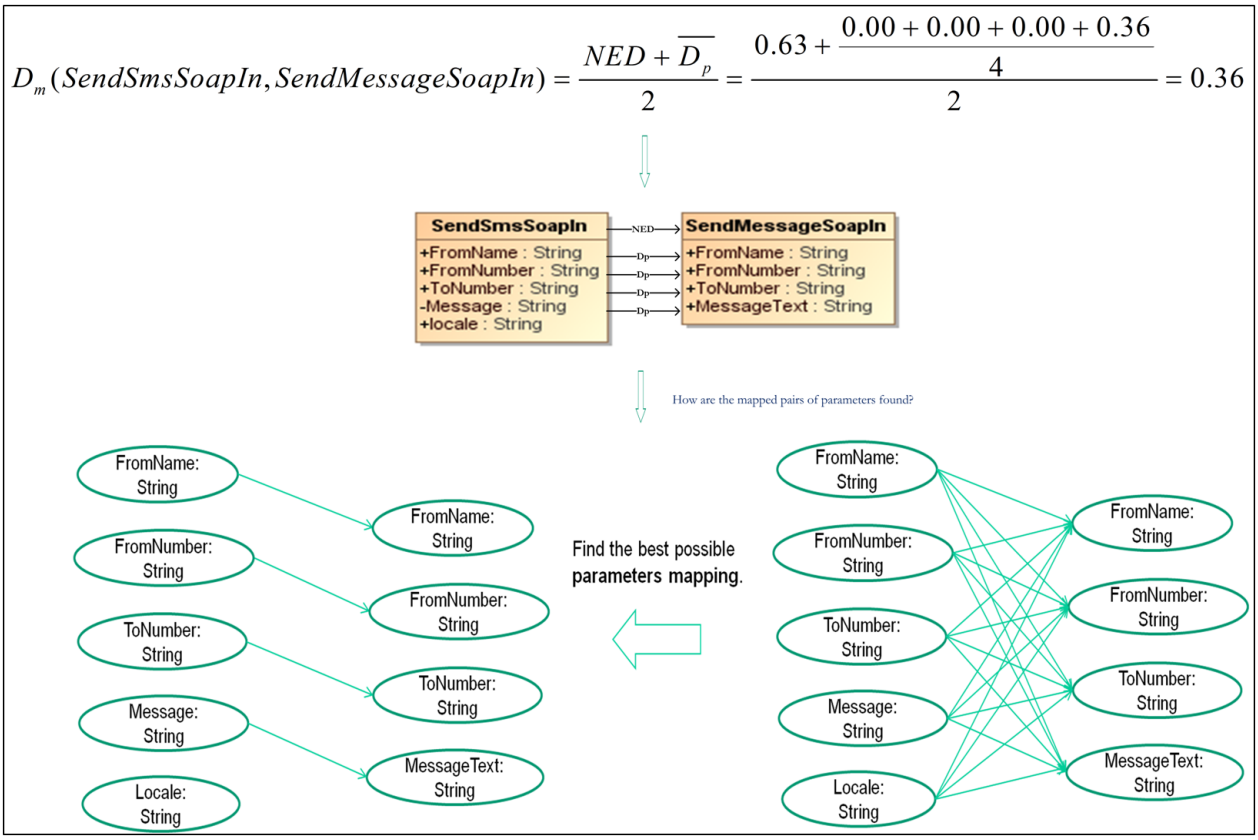


Figure 4.13: Example in calculating the distance between two messages.

for clustering. Specifically, algorithm selects the abstractions with the most similar interfaces among the ones that are currently included in the list, L (by using the function `FindCandidatePair()` in **Step 3** of Algorithm 1). In the second step, a functional abstraction, fa, is extracted that represents the aforementioned pair of abstractions; an abstract interface is defined for the new abstraction, along with a mapping from this interface to the interfaces $fa_1.i$ and $fa_2.i$ of the represented abstractions (by using the function `Extract()` in **Step 4** of Algorithm 1). This step is further detailed at the end of this subsection. In the third step, the contents of the list, L, are updated as follows: the algorithm inserts in the list the newly formed abstraction, fa, that represents the newly formed cluster and removes from the list the abstractions, fa_1 and fa_2, that represent the merged clusters (**Step 5** of Algorithm 1). Following, the contents of the distance matrix, d, are updated accordingly (by using the function `UpdateMatrix()` in **Step 6** of Algorithm 1). Finally, the newly formed functional abstraction is linked with the previously existing abstractions in the structure of the hierarchy, fh. Specifically, the descendants of the newly formed abstraction, fa, in the hierarchy, fh, are fa_1 and fa_2, while the ancestor, fa_1 and fa_2, in the hierarchy is the newly formed abstraction (**Step 7** of Algorithm 1).

The algorithm repeats the second phase until either the list, L, comprises only one functional abstraction indicating that all clusters have been merged, or no more cluster can be merged because the interfaces of the abstractions that represent these clusters are not similar at all ($D_i = 1$). After the termination of the algorithm, the functional abstractions, which remain in L, constitute the roots of the hierarchy, fh (**Step 8** of Algorithm 1).

Concerning the time complexity of Algorithm 1, in the initialization phase, the algorithm computes all the pairwise distances for the $|S|$ initial abstractions and finds the pair of abstractions with the minimum distance. Given that all the possible pairs of abstractions are $\frac{|S|*|S-1|}{2}$, the complexity of this phase is $O\left(\frac{|S|*|S-1|}{2}\right)$.

In the clustering phase, at each iteration k, the algorithm extracts a new abstraction, updates the

distances from the newly formed abstraction to every other abstractions and finds the next candidate for clustering abstractions. Taking into account that the abstractions at iteration k are $|S| - k$, the complexity of this iteration is $O(|S| - k)$. Hence, the complexity of all the iterations of the clustering phase is $O(1 + \ldots + (|S| - 1)) = O\left(\frac{|S|*(|S|-1)}{2}\right)$.

In total, the complexity of Algorithm 1 is the complexity of the initialization phase and the clustering phase, $O\left(\frac{|S|*|S-1|}{2} + \frac{|S|*|S-1|}{2}\right) = O\left(\frac{|S|*|S-1|}{2}\right)$. In other words, the complexity of the algorithm is *quadratic* with respect to the number of the services.

Algorithm 1: Producing a hierarchy of functional abstractions.

Input: a set of services, $S = \{si_1, \ldots, si_{|S|} \mid si_j : Service\}$.
Output: a hierarchy of functional abstractions, $fh : FunctionalHierarchy$.

Begin

//**Initialization phase.**
//*Initialize the list of functional abstractions,* L. **Step 1**
foreach $si_j.i \in S$ **do**
    $fa_j.i \leftarrow si_j.i$;
    $L.Insert(fa_j)$;
//*Initialize the two-dimensional distance matrix,* d. **Step 2**
foreach $(fa_m, fa_n) \in L \times L \mid (m \neq n) \wedge (fa_n, fa_m) \notin L \times L$ **do**
    $d[m][n] \leftarrow D_i(fa_m.i, fa_n.i)$;

//**Clustering interfaces and extracting abstractions phase.**
repeat
    //*Find the candidate for clustering interfaces,* $(fa_1.i, fa_2.i)$. **Step 3**
    $(min, fa_1.i, fa_2.i) \leftarrow FindCandidatePair(d, L)$;
    //*If there is at least one pair of similar interfaces, then ...*
    if $min < 1$ **then**
        //*Extract the functional abstraction,* fa. **Step 4**
        $fa \leftarrow Extract(fa_1, fa_2)$;
        //*Update the contents of the list,* L. **Step 5**
        $L.Remove(fa_1)$;
        $L.Remove(fa_2)$;
        $L.Insert(fa)$;
        //*Update the distance matrix,* d. **Step 6**
        $d \leftarrow UpdateMatrix(d, L, fa.i)$;
        //*Linking the abstraction,* fa, *with the pair* (fa_1, fa_2). **Step 7**
        $fa.desc \leftarrow \{fa_1, fa_2\}$;
        $fa_1.anc \leftarrow fa$;
        $fa_2.anc \leftarrow fa$;
until $|L| = 1$ *or* $min = 1$;
//*Forming the roots of the hierarchy,* fh. **Step 8**
foreach $fa_i \in L$ **do** $fh.root.Insert(fa_i)$;
End

Extracting an abstract interface: Extracting an abstract interface, $fa.i$, that represents two interfaces, i_1 and i_2, amounts to defining the constituents of the abstract interface along with the mapping

$fa.M$ of the abstract interface to i_1 and i_2. In further detail, the aforementioned information is determined as follows:

- A simple way to name the extracted interface is by using the *maximum common substring* of the names of $i_1.n$ and $i_2.n$.
- When calculating the distance between i_1 and i_2 the most similar pairs of operations and parameters are identified.
  - From each pair of operations, op_1, op_2 the extraction process produces an abstract operation op and corresponding operation mappings between op and the op_1, op_2. The name of the abstract operation follows the same convention as the name of the abstract interface.
  - From each pair of input/output parameters, p_1, p_2 the extraction process produces an abstract parameter p and corresponding mappings between p and p_1, p_2. The input/output parameters of the abstract operation are named by following the same convention as before. The types of the input/output parameters are defined, such that the contra-variance/co-variance rules hold.

To illustrate the extraction of an abstract interface, we assume the `sendSmsHttpPost` and the `SMSTextMessagingSoap` interfaces used in the previous example. The extracted abstract interface is named `SMS`, which is the maximum common substring of the names of these interfaces. Concerning the operations of the abstract interface, the algorithm needs the pairs of the most similar (mapped) operations of the interfaces. Having previously calculated the value of the distance of these interfaces, only one such pair of operations has been determined (i.e. (`SendSms`, `SendMessage`)). From this pair, an abstract operation is extracted; it is named `Send` following the same convention as before. Moreover, while calculating the distance of these operations, the following pairs of the most similar parameters of the input messages of these operations are identified:

- (`FromName:String`, `FromName:String`)
- (`FromNumber:String`, `FromNumber:String`)
- (`ToNumber:String`, `ToNumber:String`)
- (`Message:String`, `MessageText:String`)

To define the input message of the abstract operation, a parameter is extracted from each pair of the mapped input parameters. Each parameter is named by using the previous convention while its type is the most generic of the types of the mapped parameters. The produced parameters are the following:

- `FromName:String`
- `FromNumber:String`
- `ToNumber:String`
- `Message:String`

The output message of the abstract operation is empty because the mapping between the output messages of the operations is trivial. The extracted abstract interface is depicted in Figure 4.14. Note that to simplify the Figure, we detail only the signature of the abstract operation.

Going one step further towards a preliminary assessment of the approach we used as input to the algorithm a small but real-world set of services (Table 4.3). The general purpose of the functionality offered by these service interfaces is related to communication performed via SMS. The SMS input set

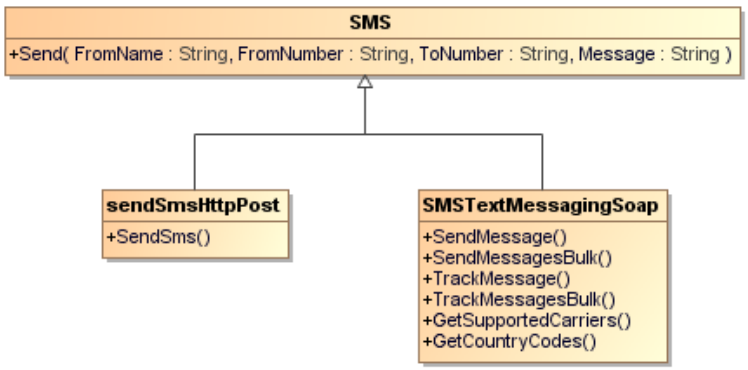


Figure 4.14: Example in extracting an abstract interface from two service interfaces.

Table 4.3: Input service sets.

Set	#services/#interfaces	Description
SMS	6/14	sending SMSs
Email	8/16	calendar services
WHOIS	6/14	find info for people
Content	26/41	weather, news services
Utilities	19/25	math, search engines, etc.

has been taken from the Woogle [DHM+04] service sets[7]. As detailed earlier, the algorithm starts from these interfaces and progressively, extracts an abstract interface from a pair of (service or abstract) interfaces until only one interface remains. The algorithm outputs a hierarchy of abstract interfaces depicted in Figure 4.15.

To investigate the capability of the algorithm in finding *useful* abstractions, we used the algorithm in further Woogle service sets. A brief description of each set of services is given in Table 4.3. By the term useful we mean abstractions that actually represent semantically compatible services. To verify the usefulness of the produced abstractions, for each set, we manually inspected the output of the algorithm and measured the percentages of the useful and useless abstractions. The results are summarized in Figure 4.16. Overall, for all input sets the proposed approach produced relatively high percentages of useful abstractions (ranging from 70% to 100%). However, in certain cases there is also a notable percentage of useless abstractions. This result was expected because the initial design of the algorithm essentially tries to maximize the amount of abstractions that can be constructed from a given set of available services.

4.2.2. Organizing services into Hierarchies of Non-functional Abstractions

In this subsection, we describe an algorithm that organizes services in hierarchies of non-functional abstractions. As discussed in subsection 4.1.2, by default we assume that the services provide similar functional properties, i.e. they belong to a particular functional abstraction. The algorithm systematically extracts non-functional abstractions, which represent groups of services that offer similar non-functional properties. Examples of such properties are provided in *the ontology of quality requirements on services* presented in Appendix A and published in [ZLHM11]. Note that this ontology does not provide a closed list of quality properties but it is defined in such a way that allows its extension with newly provided properties.

The identification of services that offer similar non-functional properties can be realized by comparing

[7]The interested reader may find the input sets at http://www.cs.uoi.gr/∼dathanas/links/Woogle.zip

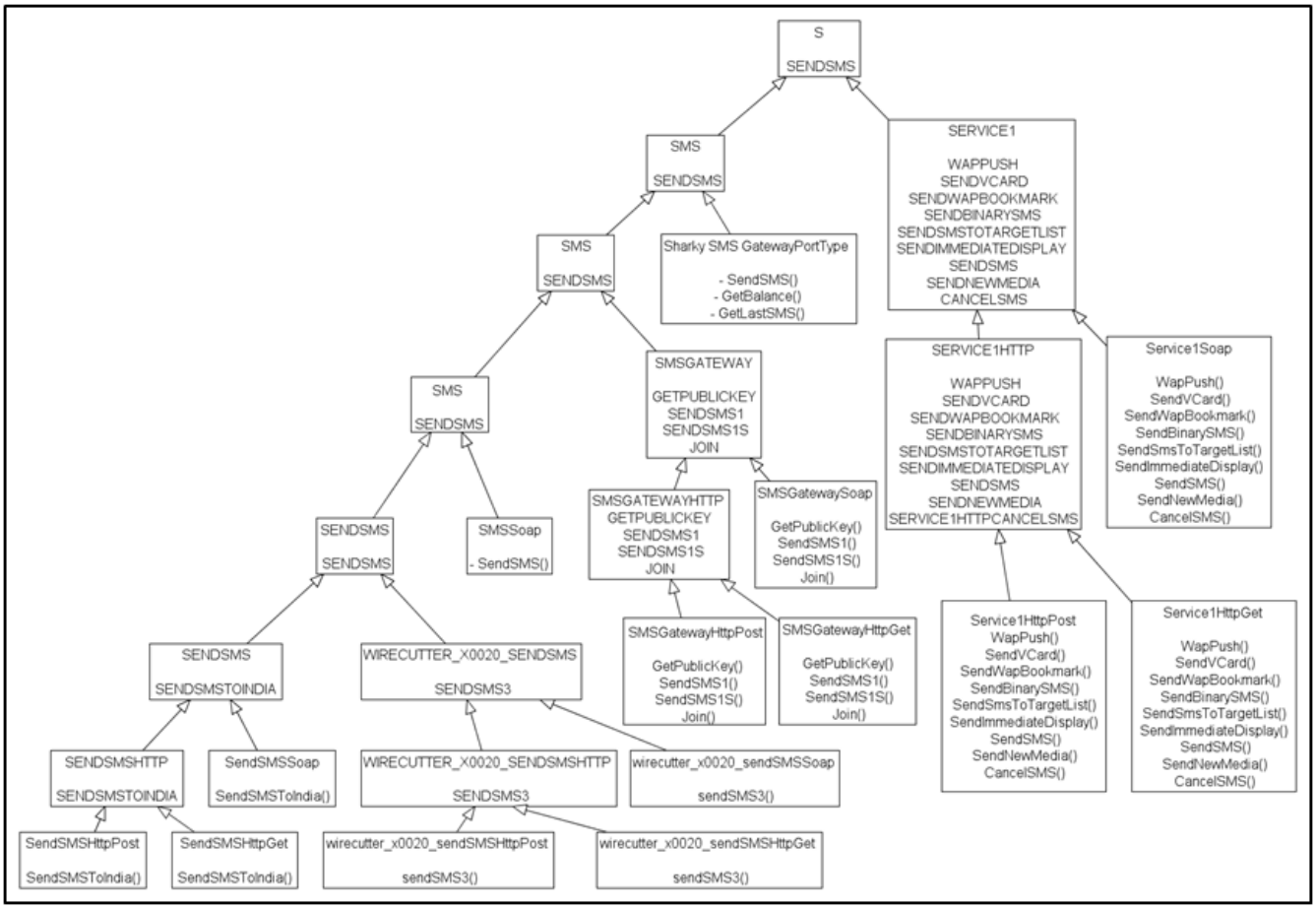


Figure 4.15: Extracted hierarchy of functional abstractions for the SMS input set.

the values of corresponding properties. In practice, these values may be discrete or continuous *numeric/quantitative* (e.g. response time, availability etc.) or *categorical/qualitative* (e.g. security, privacy etc.). In general, categorical values can also be mapped to a set of discrete numeric values in a boolean manner (they are satisfied or not) or by defining an enumeration (more than two values). For the sake of a unified approach and without loss of generality, in the remainder of this subsection, we model the non-functional properties of services as a set of numeric properties assuming that categorical properties have previously been transformed into corresponding numeric ones.

Basic concepts

According to the deliverable CHOReOS D1.3 [CHO11d], a service, si, is characterized by a non-functional description, $si.nf$ (Definition 4.23). The non-functional description is defined as a set of two tuples: $nf = \{Q, V\}$ (note that we slightly simplified the notation used in D1.3 to ease the presentation of the algorithm). The first tuple, Q, consists of a set of quality properties (Definition 4.24). Each property, $q_i \in Q$, is associated to its own *space/domain of values*, $dom(q_i)$. The second tuple, V, comprises a set of values that correspond to the quality properties (Definition 4.25). Each property, $q_i \in Q$, is characterized by the value, $v_i \in V$. Moreover, in D1.3, we defined a non-functional abstraction, nfa, in terms of a non-functional description, $nfa.nf$, and a set of services, $nfa.R$, represented by the abstraction (Definition 4.26).

In line with these definitions, we propose an algorithm that constructs groups of services based on their non-functional descriptions and abstractions that represent these groups of services. A simple way of defining the non-functional description of an abstraction is by using the average, median or range of the properties that characterize the represented services. Taking an example, assume the following

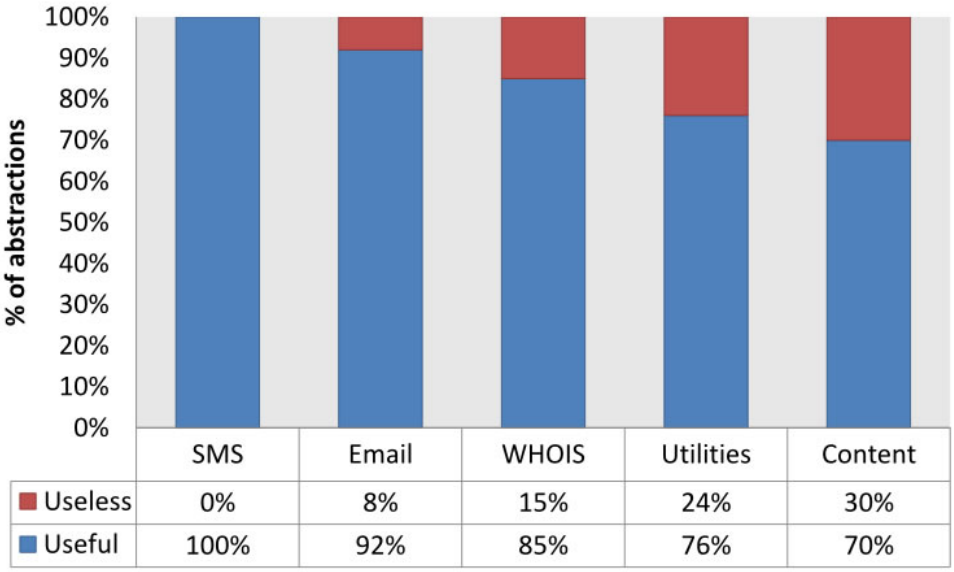


Figure 4.16: Percentages of the useful functional abstractions produced for five service sets.

$$[Non-functional\ description]\ \ nf = \{Q,\ V\} \tag{4.23}$$
$$[Quality\ properties]\ \ Q = (q_1,\ \ldots,\ q_{|Q|}) \tag{4.24}$$
$$[Values\ of\ quality\ properties]\ \ V = (v_1,\ \ldots,\ v_{|V|}):\ v_i \in dom(q_i)\ \wedge\ |V| = |Q| \tag{4.25}$$

Figure 4.17: Definition of the non-functional description of a service.

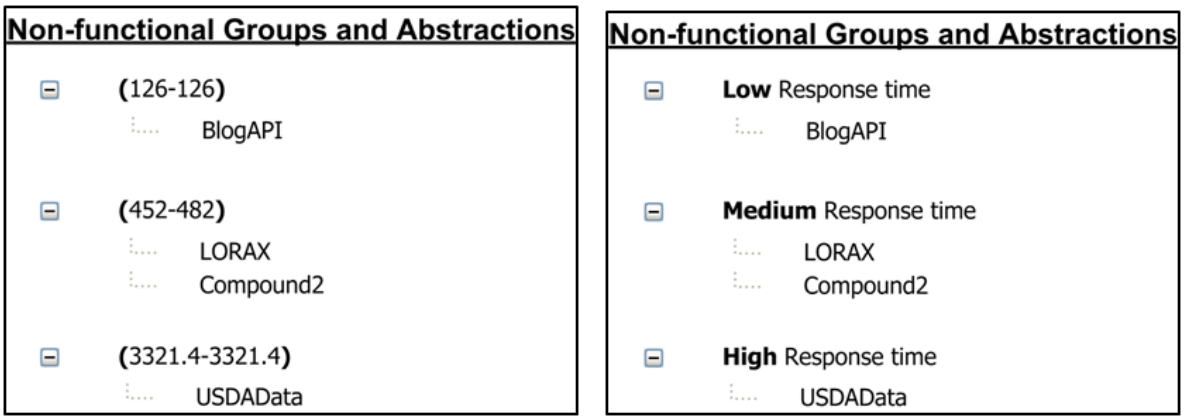
real-world services[8]: `BlogAPI`[9] , `LORAX`[10] , `Compound2`[11] and `USDAData`[12]. Furthermore, assume that these services are characterized by only one quality property, the *Response time*, the values of which are given in milliseconds in Figure 4.18.

Figure 4.18: Values of the `Response time` property for four real-world services.

In this example, we can organize similar values in three groups and define an abstraction for each group characterized by the ranges of the values of the group (Figure 4.19(a)). Presenting these abstractions to consumers enables them to search for services by browsing the abstractions and selecting groups of services via comparing the ranges of the abstractions that represent them.

However, this way of presenting abstractions is not *user-friendly*. In practice, consumers are not often interested in browsing concrete values but *user-intuitive characterizations of these values*. For more details about such characterizations see Chapter 3 of this document. Therefore, concerning the presentation of abstractions, we move one step further from simply showing concrete values. We aim at representing the values of a property by a set of *characterizations/qualifiers* such that each qualifier

[8] These services were indicatively selected from the real-world QWS dataset [AMM08].
[9] http://www.openlinksw.com/BlogAPI/services.wsdl
[10] http://lex.thelearningfederation.edu.au/WebServices/LearningExchange.asmx?WSDL
[11] http://www.mssoapinterop.org/asmx/WSDL/compound2.wsdl
[12] http://www.strikeiron.com/webservices/usdadata.asmx?wsdl



(a) Presenting abstractions in terms of concrete values. (b) Presenting abstractions in terms of qualifiers.

Figure 4.19: Alternative ways of presenting non-functional abstractions.

corresponds to a range of values. As discussed in D1.3, the qualifiers used for a quality property belong to a corresponding *abstract domain* of this property.

Returning to the previous example, assume that consumers represent the values of the `Response time` property by using the following three qualifiers: `Low`, `Medium` and `High` (Figure 4.19(b)). Consequently, three groups of values are produced and three abstractions are defined. The abstractions are characterized by the three qualifiers. In this case, domain experts browse these abstractions, instead of the ones characterized by concrete values.

Our choice of using qualifiers leads to not only a more *user-friendly* way of presenting abstractions but also a more *effective* way of searching for services. The improvement of the effectiveness is based on the fact that abstract characterizations *qualify the magnitude of the concrete values*. Specifically, while browsing concrete values, domain experts are not aware of their importance with respect to the other values. For instance, consider a set of a large number of values that is being browsed by a domain expert. He may select a specific value without being able to know whether this value is a totally good, medium or bad one. Presenting these values based on qualifiers, the domain expert can be sure about his selection.

Producing hierarchies of non-functional abstractions

The proposed algorithm consists of two parts: the first part constructs groups of non-functionally similar services along with the abstractions that represent these groups and, the second part produces a hierarchy of such abstractions. Starting with the first part, we firstly specify the required input.

Input information to the first part of the algorithm: The input of the algorithm (**Input (a), (b)** of Algorithm 2) includes a set of services. Moreover, taking into account that the abstractions are going to be produced in such a way that permits their presentation in terms of user-intuitive qualifiers, the input information further includes a set of quality properties and the abstract domains of these properties. More formally, the input is specified as follows.

a) A set of services, $S = \{si_1, \ldots, si_{|S|}\}$.

b) A set of tuples that consist of quality properties and abstract domains, $L = \{(q_i, \delta_i) \mid q_i \in \bigcup_{\forall si_j.nf \in S} (si_j.nf.Q)\}$. In particular, each pair, (q_i, δ_i), provides the abstract domain, δ_i of the property, q_i. Note that the abstract domain, δ_i, of the property, q_i, is defined as a tuple of qualifiers, $\delta_i = (qf_{i_1}, \ldots, qf_{i_{|\delta_i|}})$.

Pre-processing the input: The input of the algorithm may need pre-processing in case the non-functional descriptions do not include the same quality properties. This pre-processing, which is implemented by **Step 1** of Algorithm 2, is required because the non-functional abstractions are produced over a common set of properties. Therefore, the first step of the pre-processing is to determine the set of the properties, Q', which are common in the descriptions of all the input services, $Q' = \bigcap_{\forall\ si_j\ \in\ S}(si_j.nf.Q)$ (by using the function `FindCommonProperties()` in **Step 1** of Algorithm 2). Following, the pre-processing produces a filtered version of the non-functional descriptions of each one of the input services as follows.

- For the non-functional description, $si_j.nf$, of each service, $si_j \in S$, the quality properties of the description, $si_j.nf.Q$, are filtered to keep the properties that are common in the descriptions of all services, $si_j.nf.Q \leftarrow Q'$.
- For the non-functional description, $si_j.nf$, of each service, $si_j \in S$, the set of values of the description, $si_j.nf.V$, is filtered to keep only the values that correspond to the common properties: $v_i \in si_j.nf.V \Leftrightarrow q_i \in Q'$ (function `FilterServiceDescriptions()` in **Step 1** of Algorithm 2).
- The pairs of the input set, L, are filtered to produce the set, L', that keeps only the pairs that involve the common properties of Q', namely, $(q_i,\ \delta_i) \in L' \Leftrightarrow q_i \in Q'$ (function `FilterAbstractDomains()` in **Step 1** of Algorithm 2).

After performing the pre-processing, note that the set of quality properties, $si_j.nf.Q$, of the non-functional description, $si_j.nf$, of a service, $si_j \in S$, may be empty because the service may not provide values for any of the common properties, Q'. Therefore, during the pre-processing, the input set, S, is also filtered to produce the set of services, S', that provide values for the common properties: $si_j \in S' \Leftrightarrow si_j.nf.Q \neq \emptyset$.

Construction of non-functional groups of services and abstractions: Having performed the pre-processing of the input, the algorithm turns to the construction of groups of services that provide similar quality properties along with abstractions that represent these groups. First, the number of the groups/abstractions is calculated based on the input of the algorithm. Specifically, this number depends on the cardinalities of the abstract domains of the common properties. In the case of only one property, as revealed by Figure 4.19(b), the groups/abstractions are as many as the qualifiers of the abstract domain of the `Response time` property (δ = (`Low`, `Medium`, `High`)). In the case of two or more properties, the number of the groups/abstractions is equal to the different combinations of the qualifiers of each property. Taking an example with two properties characterized by the abstract domains, δ_1 = (`Low`, `High`) and δ_2 = (`Low`, `Medium`, `High`) respectively, the number of the groups equals to six ($|\delta_1| * |\delta_2| = 2 * 3$).

After calculating the number of the groups/abstractions, the algorithm creates these groups of services and defines the corresponding abstractions. To produce the groups, the algorithm compares the quality values of the services, S', for their common properties, Q'. Each produced group of services is characterized by a range of values for each property. An abstraction for each group of services is defined as follows: for each property, the range of values is associated to a suitable qualifier; the set of these qualifiers constitutes the non-functional description of the abstraction.

The typical way for constructing groups of services that provide similar quality properties, is using a clustering method. Such an approach has already been proposed in [MBK+09]. This approach models each service as a tuple/vector of normalized quality values. Given such vectors, this approach uses a partitional clustering method, *K-means* [Llo82], in order to group services characterized by similar vectors. In our problem we also use K-means because it is the most usual method for producing a fixed number of clusters.

However, the application of clustering in many dimensions/properties may induce certain problems. As the number of the dimensions grows the distance (independently of the type of metric) used by the clustering method to measure the dissimilarity among vectors becomes less precise. As a result,

the identification of the most and the least similar vectors to form well-separated clusters of vectors becomes meaningless. Moreover, in a high number of dimensions, several of these dimensions are not usually meaningful for forming clusters. In the literature, this phenomenon is called *local feature relevance*, in which the *correlation/irrelevance* of the values of a subset of the dimensions influences the creation of clusters [KKZ09].

A common suggestion to overcome this problem is to use *dimensionality reduction* techniques that select a subset of the dimensions over which the clustering is performed. However, these techniques do not always lead to good clustering results because the selection of the subset of the dimensions is performed in a global way. On the other hand, in practice, good clusters may be found based on different combinations of the dimensions. However, the optimal subsets of the dimensions should be searched into an exponential number of combinations leading to computationally infeasible solutions. To shrink down the complexity of these solutions, various heuristics and assumptions are adopted by approaches called *subspace clustering* approaches [PHL04, KKZ09].

The *subspace clustering* approaches can be divided into two types, the *top-down* and the *bottom-up* approaches. The former starts from the set of all the dimensions and splits them in maximal subsets. The latter starts from each dimension and merges them in maximal subsets. We use a *bottom-up subspace clustering* approach that starts from each dimension independently to deal with the multi-dimensionality problem. In particular, our approach comprises two phases. In the first phase, the approach produces groups of services for each property that are henceforth called *property-based* groups. In the second phase, the approach combines the *property-based* clusters to form high-dimensional clusters called *joint* groups of services.

a) **Production of *property-based* groups of services**

In the first phase, the approach deals with the values provided by each property independently. Specifically, the approach organizes the values provided by the services into as many groups as the qualifiers. In further detail, K-means is applied for each property, $q_i \in Q'$, as follows (by using the function `ProducePropertyBasedGroups()` in **Step 5** of Algorithm 2).

Provided information to K-means:

- the abstract domain, $\delta_i = (qf_{i_1}, \ldots, qf_{i_{|\delta_i|}})$, of the property, q_i
- the values of the property, q_i, provided by the non-functional descriptions of the input services, S'

Steps for producing property-based groups of services by K-means:

a) The input values are sorted in an ascending order producing the tuple, $(v_{i_1}, \ldots, v_{i_{|S'|}})$.

b) The sorted input values, $(v_{i_1}, \ldots, v_{i_{|S'|}})$, are organized into a set of $|\delta_i|$ clusters, $\{C_{i_1}, \ldots, C_{i_{|\delta_i|}}\}$. Note that several of the clusters may be empty depending on the distribution of the values.

c) For each cluster, C_{i_w}, the minimum, v_{i_k}, and the maximum value, v_{i_m}, are determined defining a range of values, $R_{i_w} = (v_{i_k}, v_{i_m}) : v_{i_k} \leq v_{i_m} \wedge 1 \leq k \leq m \leq |S'| \wedge 1 \leq w \leq |\delta_i|$.

d) The ranges of the clusters are sorted in an ascending order with respect their endpoints producing a tuple of ranges, $R_i = (R_{i_1}, \ldots, R_{i_{|\delta_i|}}) : R_{i_1} = (v_{i_1}, v_{i_t}) \wedge R_{i_w} = (v_{i_k}, v_{i_m}) \wedge R_{i_{w+1}} = (v_{i_n}, v_{i_o}) \wedge R_{i_{|\delta_i|}} = (v_{i_u}, v_{i_{|S'|}}) \wedge 1 \leq w \leq |\delta_i|-1 \wedge v_{i_t} \leq v_{i_k} \wedge v_{i_m} \leq v_{i_n} \wedge v_{i_o} \leq v_{i_u}$.

e) Given that the ranges have been sorted, the first range, R_{i_1} corresponds to the first qualifier, qf_{i_1}, of the abstract domain δ_i, the second range, R_{i_2} to the second qualifier, qf_{i_2}, and so on. Therefore, each range, R_{i_w}, is characterized by the corresponding qualifier, $qf_{i_w} \in \delta_i$.

f) For each range, R_{i_w}, the group of services, $S_{i_w} \subseteq S'$, that provide values in this range is determined. In total, for each property, a tuple of groups of services is defined, $S_i = (S_{i_1}, \ldots, S_{i_{|\delta_i|}})$.

To illustrate the production of property-based groups, we present an example in which the algorithm accepts as input the non-functional descriptions of 50 real-world services that provide values for two properties, $Q' = \{q_1,\ q_2\} = \{Response\ time,\ Availability\}$[13]. In addition, the algorithm accepts as input the following abstract domains for each property:

- L' = (q_1, δ_1), (q_2, δ_2) where,
  - (q_1, δ_1) = (`Response time`, (`Low`, `High`))
  - (q_2, δ_2) = (`Availability`, (`Low`, `Medium`, `High`))

The K-means is applied for each property, dividing the values of the `Response time` property into two clusters characterized by a tuple of two ranges whereas, the values of the `Availability` property are divided into three clusters characterized by a tuple of three ranges. In particular, these two tuples of ranges are following:

i) R_1 = (R_{1_1}, R_{1_2}) = ((49.4, 1360), (3321.4, 3321.4))
ii) R_2 = (R_{2_1}, R_{2_2}, R_{2_3}) = ((18, 19), (36, 56), (70, 100))

Concerning the defined ranges for the values of the `Response time` property, R_{1_1}, corresponds to the qualifier Low, whereas, R_{1_2}, corresponds to the qualifier $High$. Similarly, for the `Availability` property, R_{2_1}, corresponds to the qualifier Low, R_{2_1}, to $Medium$, and, R_{3_1}, to $High$.

Finally, for the `Response time` property, R_{1_1}, characterizes a group of 49 services, $|S_{1_1}| = 49$, whereas, R_{1_2}, characterizes only one service, $|S_{1_2}| = 1$ (Figure 4.20).

For the `Availability` property, R_{2_1}, characterizes a group of two services, $|S_{2_1}| = 2$, R_{2_2}, a group of five services, $|S_{2_2}| = 5$, and, R_{2_3}, a group of 42 services, $|S_{2_3}| = 42$ (Figure 4.21).

b) **Production of *joint* groups of services**

A joint group is formed by taking a combination of property-based groups, one for each property. The services that belong to a joint group is found by intersecting the services of its constituent property-based groups. The ranges of values that characterize the services of a joint group are the ones that characterize its constituent property-based groups. To form the joint groups, the algorithm calculates all the possible combinations of ranges that characterize the property-based groups via finding the *Cartesian product*, $R = R_1 \times \ldots \times R_{|Q'|}$ (by using the function `ProduceJointGroups()` in **Step 6** of Algorithm 2). More specifically:

a) Each combination of ranges is a tuple, $JR_w = (R_{1_k},\ \ldots,\ R_{|Q'|_m}) : R_{i_j} \in R_i\ \wedge\ 1 \leq i \leq |Q'|\ \wedge\ 1 \leq k \leq |\delta_1|\ \wedge\ 1 \leq m \leq |\delta_{|Q'|}|\ \wedge\ 1 \leq w \leq |R|$.
b) For each combination of ranges, JR_w, the algorithm constructs the joint group of services, JS_w, whose quality values belong to these ranges.
c) Each combination of ranges, JR_w, is characterized by a tuple of qualifiers, one for each property, $JQF_w = (qf_{1_k},\ \ldots,\ qf_{|Q'|_m}) : qf_{i_j} \in \delta_i\ \wedge\ 1 \leq i \leq |Q'|\ \wedge\ 1 \leq k \leq |\delta_1|\ \wedge\ 1 \leq m \leq |\delta_{|Q'|}|\ \wedge\ 1 \leq w \leq |R|$.
d) In total, the produced joint groups of services, $JS = \{JS_1,\ \ldots,\ JS_{|R|}\}$, are characterized by the qualifiers, $JQF = \{JQF_1,\ \ldots,\ JQF_{|R|}\}$, that corresponds to the ranges, $JR = \{JR_1,\ \ldots,\ JR_{|R|}\}$.

To illustrate the production of joint groups, we return to the previous example. The algorithm computes the Cartesian product, $R = R_1 \times R_2$, of the ranges, R_1 and R_2, of the produced property-based groups of services. As a result of the Cartesian product, we have the following combinations/tuples of ranges:

[13] In the example, we used the first 50 services out of the 2507 available services of the QWS dataset [AMM08].

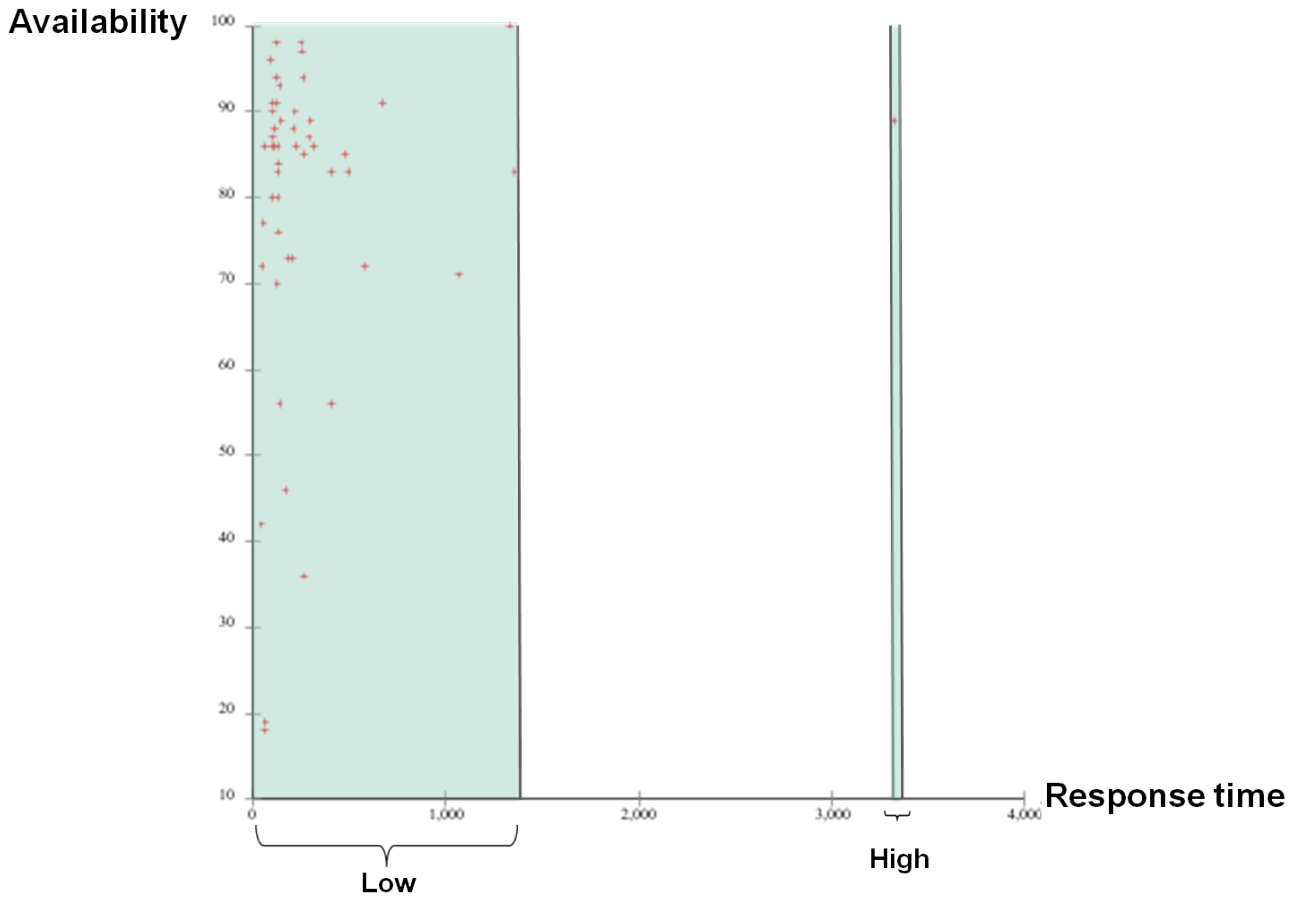


Figure 4.20: Example in producing property-based groups for 50 services based on the values that they provide for the `Response time` property.

i) JR_1 = (R_{1_1}, R_{2_1}) = ((49.4, 1360), (18, 19))
ii) JR_2 = (R_{1_1}, R_{2_2}) = ((49.4, 1360), (36, 56))
iii) JR_3 = (R_{1_1}, R_{2_3}) = ((49.4, 1360), (70, 100))
iv) JR_4 = (R_{1_2}, R_{2_1}) = ((3321.4, 3321.4), (18, 19))
v) JR_5 = (R_{1_2}, R_{2_2}) = ((3321.4, 3321.4), (36, 56))
vi) JR_6 = (R_{1_2}, R_{2_3}) = ((3321.4, 3321.4), (70, 100))

Each combination of ranges corresponds to a joint group. The services that belong to such a group are found by intersecting the services of its constituent property-based groups. Specifically, the six joint groups of services are defined as follows (we have also calculated the number of the services contained in these groups).

i) $JS_1 = \{S_{1_1} \cap S_{2_1}\} \Rightarrow |JS_1| = 2$
ii) $JS_2 = \{S_{1_1} \cap S_{2_2}\} \Rightarrow |JS_2| = 5$
iii) $JS_3 = \{S_{1_1} \cap S_{2_3}\} \Rightarrow |JS_3| = 42$
iv) $JS_4 = \{S_{1_2} \cap S_{2_1}\} \Rightarrow |JS_4| = 0$
v) $JS_5 = \{S_{1_2} \cap S_{2_2}\} \Rightarrow |JS_5| = 0$
vi) $JS_6 = \{S_{1_2} \cap S_{2_3}\} \Rightarrow |JS_6| = 1$

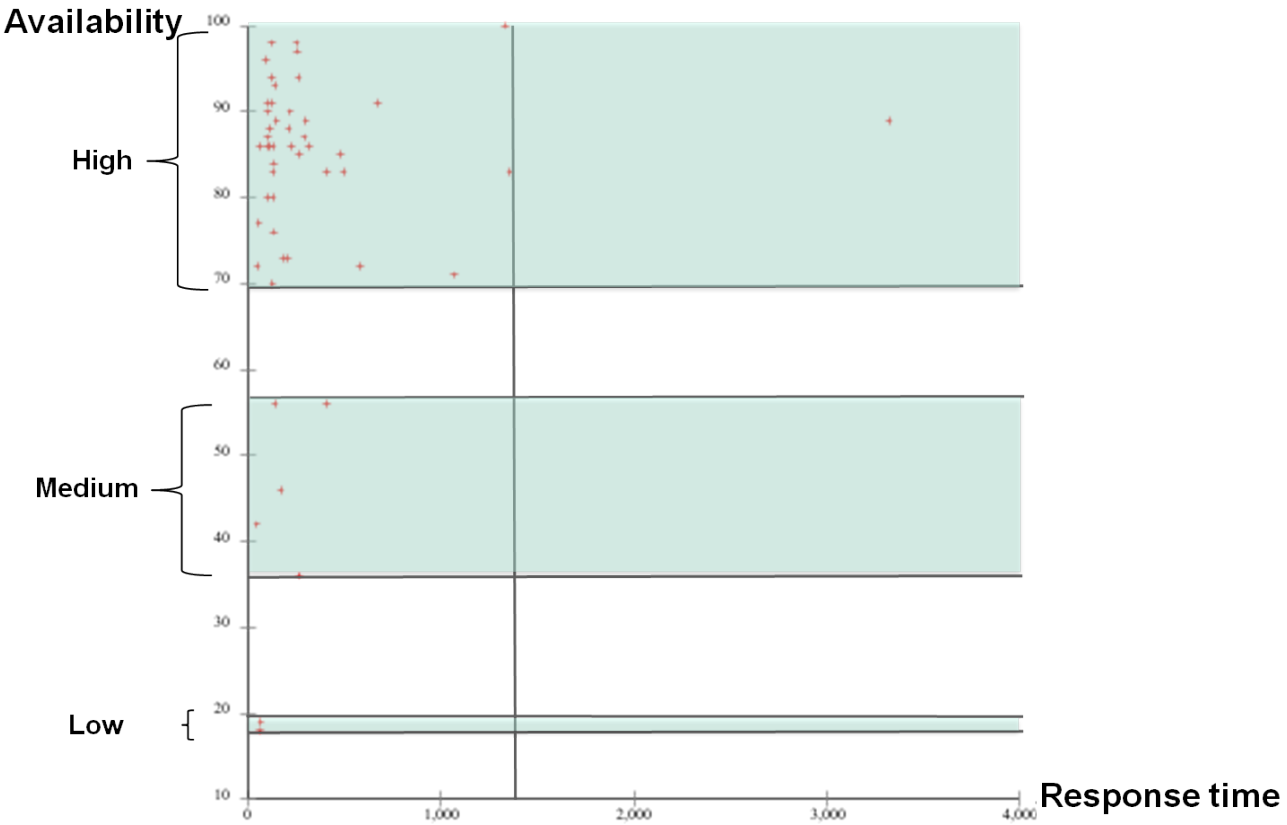


Figure 4.21: Example in producing property-based groups for 50 services based on the values that they provide for the `Availability` property.

Based on the two previous phases of the algorithm and Definition 4.26, for each one of the previously produced joint groups of services, JS_w, the algorithm extracts an abstraction, nfa, as follows (**Step 7** of Algorithm 2):

- The set of the represented services, $nfa.S$, is JS_w ($nfa.R \leftarrow JS_w$).
- Concerning the non-functional description, $nfa.nf$ (Definition 4.27),
  - the tuple of the quality properties, $nf.Q$ (Definition 4.28), consists of the properties for which all the services provide values ($nf.Q \leftarrow Q'$),
  - the tuple of the qualifiers, $nf.V$ (Definition 4.29), comprises the qualifiers, JQ_w, that characterize the joint group, JS_w,
  - the mapping, α (Definition 4.30), from the qualifiers, JQ_w, to the ranges, JR_w, is defined such that for $qf_{i_j} \in JQF_w$, $q_i \in JQ_w$ and $R_{i_j} \in JR_w$, we have $\alpha(q_i,\ qf_{i_j}) = R_{i_j}$.

Based on these definitions, in the previous example, the algorithm extracts an abstraction for each one of the six joint groups as follows. Note that all these abstractions are characterized by the common tuple of quality properties, Q = (`Response time`, `Availability`). For instance, the first abstraction, nfa_1, is characterized by the tuple of qualifiers, $nfa_1.V$ = (`Low`, `Low`), that correspond to the ranges, JR_1 and represent the joint group of services, JS_1.

Construction of a hierarchy of non-functional abstractions: Having produced groups of services that provide similar non-functional descriptions, an undesirable scenario may happen. Depending on the distribution of the values of properties provided by the service descriptions, several groups may

$$[Non-functional\ abstraction]\ \ nfa = \{nf,\ S,\ anc,\ desc\} \tag{4.26}$$

$$[Non-functional\ description]\ \ nf = (Q,\ V) \tag{4.27}$$

$$[Quality\ properties]\ \ Q = (q_1,\ \ldots,\ q_{|Q|}) \tag{4.28}$$

$$[Qualifiers]\ \ V = (qf_1,\ \ldots,\ qf_{|Q|}) : qf_i \in \delta_i \tag{4.29}$$

$$[Mappings\ from\ qualifiers\ to\ ranges]\ \ \alpha : nf \rightarrow JR \tag{4.30}$$

$$[Ranges\ of\ joint\ group]\ \ JR = (R_{1_k},\ \ldots,\ R_{|Q|_m}) : R_{i_j} \in R_i\ \wedge\ 1 \leq i \leq |Q|\ \wedge\ 1 \leq k \leq |\delta_1|\ \wedge\ 1 \leq m \leq |\delta_{|Q|}| \tag{4.31}$$

$$[Ranges\ of\ property-based\ group]\ \ R_i = (R_{i_1},\ \ldots,\ R_{i_{|\delta_i|}}) : R_{i_1} = (v_{i_1},\ v_{i_t})\ \wedge\ R_{i_w} = (v_{i_k},\ v_{i_m})\ \wedge\ R_{i_{w+1}} = (v_{i_n},\ v_{i_o})\ \wedge\ R_{i_{|\delta_i|}} = (v_{i_u},\ v_{i_{|S'|}})\ \wedge\ 1 \leq w \leq |\delta_i| - 1\ \wedge\ 1 \leq w \leq |\delta_i| - 1\ \wedge\ v_{i_t} \leq v_{i_k}\ \wedge\ v_{i_m} \leq v_{i_n}\ \wedge\ v_{i_o} \leq v_{i_u} \tag{4.32}$$

$$[Represented\ services]\ \ S = \{si_1,\ \ldots,\ si_{|S|}\} \tag{4.33}$$

Figure 4.22: Definition of the notion of the non-functional abstraction.

be overcrowded. This scenario complicates the task of selecting one of the services of these groups. For instance, consumers, who are in need of an appropriate service, can pick up one of the produced groups and further browse the contents of the group. If consumers select an overcrowded group, they unavoidably traverse a big number of services. In the previous example, we also faced this scenario in the case of the abstraction characterized by the qualifiers, **`Low`** and **`High`**; this abstraction represents 42 services. To avoid this scenario, we further divide overcrowded groups in smaller sub-groups deriving a hierarchy of groups and consequently, a hierarchy of abstractions.

Therefore, turning to the second part of the algorithm, the already produced groups and abstractions are further processed. In particular, the algorithm divides each joint group into smaller ones. A group is divided only if it contains more services than the value of a threshold, N (condition of division). This threshold is provided as input to the algorithm (**Input (c)** of Algorithm 2). Towards producing a hierarchy of groups, the algorithm iteratively repeats the previously detailed **Step 5–7** of Algorithm 2.

Technically, the abstractions that represent the candidate for division groups are maintained as elements of a list, $NFA = \{nfa_1,\ \ldots,\ nfa_{|NFA|} \mid nfa_i : NonFunctionalAbstraction\}$. Initially, the algorithm produces an abstraction that describes all the services. At each iteration, the algorithm removes from the list an abstraction, $nfa_i \in NFA$, and defines a set of lower-level abstractions. Each newly formed abstraction, nfa_{i_k}, is linked with the abstraction, nfa_i, from which it is derived. In other words, the abstraction, nfa_{i_k}, is descendant of the abstraction, nfa_i. To hold this information, each non-functional abstraction, nfa_{i_k}, is characterized by two extra elements: an element that denotes its descendant, $nfa_{i_k}.desc$, and one for its ancestor, $nfa_{i_k}.anc$ (Definition 4.26). Therefore, the descendants of the divided abstraction, nfa_i, are each one of the newly formed abstractions, nfa_{i_k}, whereas, the ancestor of nfa_{i_k} is the nfa_i. Overall, the algorithm repeats the **Step 5–7** until the list of candidate for division abstractions, NFA, is empty ($|NFA| = 0$).

To determine the time complexity of Algorithm 2, we firstly find the complexity of the initialization phase of the algorithm in which it performs the following three tasks:

- Finding the common quality properties of the input services (multi-set intersection).

  In this task, the algorithm firstly sorts the set of the properties provided by each service. Assuming that the maximum number of properties provided by all the services is, $p = \max_{\forall\ si_j\ \in\ S} |si_j.nf.Q|$, the sorting task for each service costs, $O(p^2)$, and for all the services $O(|S| * p^2)$. Secondly, the algorithm finds the intersection of the sets of the properties of each service as a sequence of pair-wise intersections. For a pair of two sets, the intersection is performed in $O(p)$ time [BYS10] and similarly, for the sequence of intersections of $|S| - 1$ pairs, the complexity is $O((|S| - 1) * p)$.

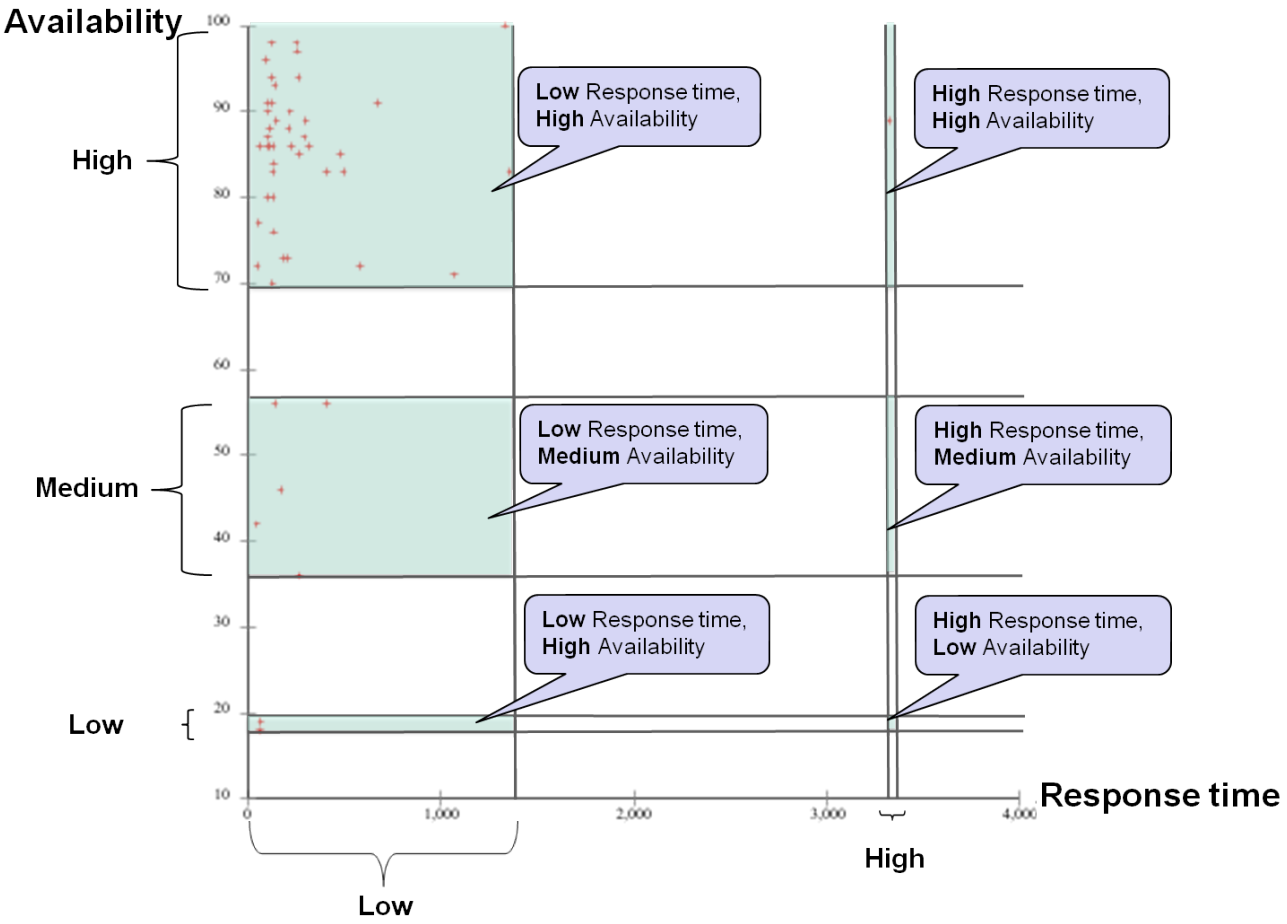


Figure 4.23: Example in producing joint groups for 50 services based on the values that they provide for two properties.

- Pre-processing the non-functional descriptions of the input services in order to filter the set of the properties and the values that they provide.

  Given that this task is performed for each service and that each service provides at most p values (one for each property) the filtering costs $O(|S| * p)$.

- Creating an abstraction that describes all the filtered services, S'.

  Taking into account that in this task a range of values is defined for each common property, the complexity is $O(p)$.

In total, the initialization phase costs at most $O(|S| * p^2 + (|S| - 1) * p + |S| * p + p) = O(|S| * p^2)$ time. In the clustering phase, at each iteration, the algorithm performs the following three tasks:

- Producing property-based groups of services.

  In this task, the algorithm applies K-means for each property, $q_i \in Q'$, in $O(I_1 * |\delta_i| * |S|)$ time [TSK06], where, I_1, is the number of the iterations required by K-means for convergence. Assuming that, d, is the maximum size of the abstract domains of the properties, $d = \max_{1 \leq i \leq |Q'|} |\delta_i|$, then the overall complexity of this task for all properties is $O(|Q'| * I_1 * d * |S|)$. Moreover, considering that the number of the iterations is constant, the complexity is $O(|Q'| * d * |S|)$.

- Producing joint groups of services.

  In this task, the algorithm finds the Cartesian product of the ranges, $R_1, \ldots, R_{|Q'|}$, of the property-based groups. Given that the number the produced combinations equals to $|R| = |R_1| * \ldots *$

$|R_{|Q'|}| = \prod_{\forall q_i \in Q'} |\delta_i|$, the complexity of this task is $O(\prod_{\forall q_i \in Q'} |\delta_i|)$.

Concerning the number of the steps of the clustering phase, it depends on: (a) the size (number of contained services) of the produced property-based groups; (b) the size of the joint groups; (c) the value of the threshold, N. If the size of these groups remains big enough during the execution of the clustering phase with respect to the threshold, N, then this phase is repeated many times. To show the worst case scenario for the size of a joint group, we start by assuming that the algorithm performs the first iteration. In the beginning of this iteration, only one joint group exists and it contains the initial number of services, $|S'|$. Next, K-means is applied for each property, q_i, to divide the $|S'|$ services in $|\delta_i|$ groups. The worst case happens when K-means produces $|\delta_i| - 1$ singleton clusters (that correspond to outliers) and one big cluster (property-based group) that contains $|S'| - (|\delta_i| - 1)$ services. We assume that K-means does not produce empty clusters. The final step of this iteration is to form joint groups. The worst case happens when all joint groups are empty or singleton except for only one group that is produced based on the previous big property-based group which contains $|S'| - (|\delta_i| - 1)$ services. Assuming that the same scenario as before happens at the second iteration, the most dense produced joint group will contain $|S'| - 2 * (|\delta_i| - 1)$ services. Therefore, the number of the iterations, I_2, needed for the algorithm to produce joint groups that all have size smaller than the threshold is $I_2 = \frac{|S'|-N}{|\delta_i|}$. Moreover, this number is even smaller when the value of the threshold is 1. Hence, the steps of the clustering phase are $O\Big(\frac{|S'|}{|\delta_i|}\Big) = O\Big(\frac{|S'|}{d}\Big)$ and the overall complexity of the clustering phase is $O\Big(\frac{|S'|}{d} * (|Q'| * d * |S|) * \prod_{\forall q_i \in Q'} |\delta_i|\Big) = O\Big(|S|^2 * |Q'| * \prod_{\forall q_i \in Q'} |\delta_i|\Big)$.

Finally, the complexity of Algorithm 2 is the complexity of the initialization phase and clustering phase, $O\Big(|S| * p^2 + |S|^2 * |Q'| * \prod_{\forall q_i \in Q'} |\delta_i|\Big) = O\Big(|S|^2 * |Q'| * \prod_{\forall q_i \in Q'} |\delta_i|\Big)$. In other words, the complexity of the algorithm is *polynomial* with respect to the number of the services, the number of the common properties and the sizes of the abstract domains of the properties.

Algorithm 2: Producing a hierarchy of non-functional abstractions.

Input:
(a) a set of services, $S = \{si_1, \ldots, si_{|S|}\}$
(b) a set of pairs of abstract domains and quality properties, $L = \{(q_i, \delta_i)\}$
(c) the maximum number of services represented by a non-functional abstraction, N.
Output: a hierarchy of non-functional abstractions, $nfh : NonFunctionalHierarchy$.

Begin

```
//Pre-process the input sets, S and L.                                   Step 1
Q' ← FindCommonProperties(S); //Multi-set intersection.
S' ← FilterServiceDescriptions(S, Q');
L' ← FilterAbstractDomains(L, Q');

//Create an abstraction, nfa, that describes all the services, S'.      Step 2
nfa.S ← S';
nfa.nf.Q ← Q';
nfa.anc ← ∅;
foreach q_i ∈ Q' do
    nfa.nf.qf_i ← "all";
    nfa.nf.JR_i ← (min(dom(q_i)), max(dom(q_i)));
    α(q_i, qf_i) ← JR_i;
nfh.root ← nfa; //Set the, nfa, as the root of the, nfh.               Step 3
NFA.Insert(nfa); //Initialize a list of abstractions, NFA.             Step 4

//Develop a hierarchy of non-functional abstractions.
repeat
    nfa_i ← NFA.Remove(); //The candidate for division abstraction.
    //If the, nfa_i, represents at least, N, services.
    if nfa_i.S > N then
        //Produce property-based groups of services.                   Step 5
        foreach q_i ∈ Q' do (S_i, R_i) ← ProducePropertyBasedGroups(q_i, δ_i, S');
        //Produce joint groups of services.                            Step 6
        (JS, JQF, JR) ← ProduceJointGroups({S_1, ..., S_|Q'|}, {R_1, ..., R_|Q'|});
        //Define an abstraction for each joint group of services.      Step 7
        foreach JS_w ∈ JS do
            nfa_iw.S ← JS_w;
            nfa_iw.Q ← Q';
            nfa_iw.anc ← nfa_i;
            nfa_i.desc ← nfa_iw;
            foreach q_i ∈ Q' do
                nfa_iw.nf.qf_i ← JQF_w;
                nfa_iw.nf.JR_i ← JR_w;
                α(q_i, JQF_w) ← JR_w;
            NFA.Insert(nfa_iw);
until |NFA| = 0;
```

End

5 Analysis of adaptable QoS-aware ULS choreographies

In this chapter, we describe the *Design and run-time analysis* activity of the CHOReOS dynamic process (Figure 2.1). Such activity includes a theoretical model for *analyzing and predicting scalability and associated QoS parameters* for computation in large-scale choreographies (Section 5.1). The design and run-time activity also encompasses the *choreography stability and interdependency analysis* (Section 5.2), which provides the foundation for a series of maintenance and refactoring tasks. For now, we employ this kind of analysis to support *change impact assessment*, which is a crucial step towards improving choreography evolvability.

5.1. QoS prediction model

The core of scalability analysis depends on QoS prediction. In an abstract way indeed, the scalability of a computation refers to the impact on its QoS when we make scale some variables of on which the computation depends. In this part we will show how we envision to compute choreography QoS parameters first. We propose a model that we consider as a first stage for a general model for scalability analysis.

The QoS prediction model serves at design-time for choreography designer to evaluate the quality of a composition of services. We propose here a model that giving in input a set of BPMN constructs defining a choreography and QoS dimensions (Service Response Time, Availability, Capacity) can estimate on the different dimensions the QoS of the choreography. For now, the proposed model considers a subset of BPMN constructs among all the available ones. However, the model is generic enough for future extension.

Our QoS prediction model targets the construction of a QoS component that will be used as follows: From the choreography designer, the QoS component receives a BPMN choreography. This choreography is specified with a set of parameters values associated to activities that it comprises (we explain later details about choreography specification). The QoS component processes the submitted inputs

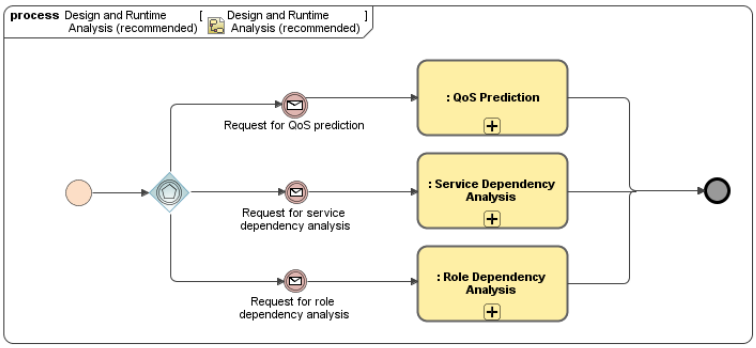


Figure 5.1: Design and run-time analysis process

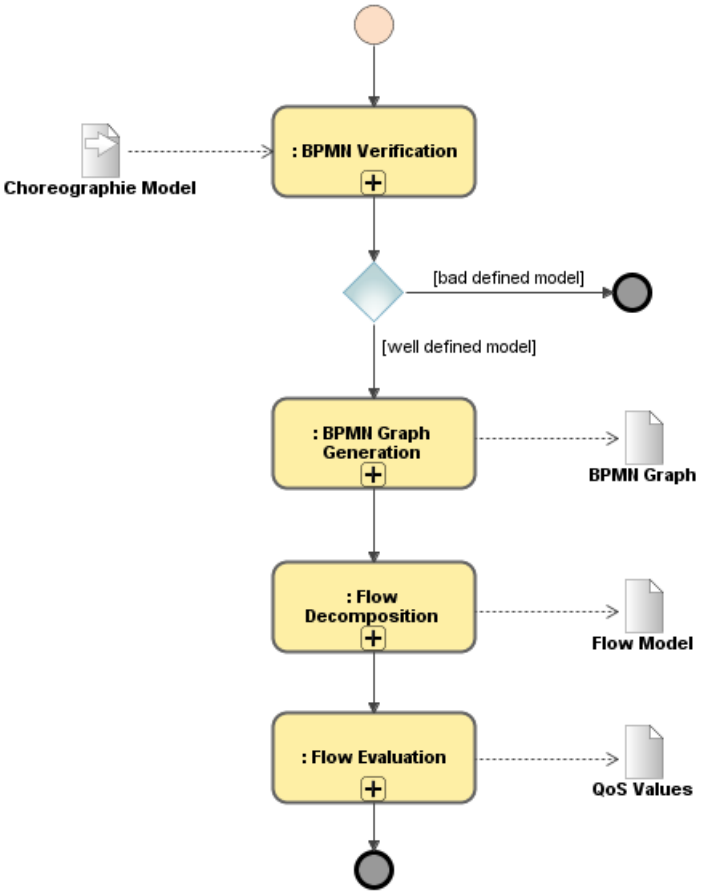


Figure 5.2: Process of Choreography QoS evaluation.

and provides a result on the choreography QoS parameters. In CHOReOS, multiple QoS parameters can be considered for measuring QoS. We first restricted ourselves to *Service Response Time* that is the mean time expected for choreography execution. However, the methodology employed for this case can be easily extended to other QoS dimensions.

For the computation of QoS value on different dimension (QoS parameters), we propose a methodology whose key steps are described on Figure 5.2. A BPMN choreography is submitted by the choreography designer. This BPMN describes interactions between activities constitutive of the choreography and associate to each activity a set of known QoS values. From it, we construct an internal graph associated to the choreography. The choreography is then decomposed in flows, which are evaluated (using inputs QoS values). In the following, we explain the main components and steps required for this methodology. From now, we will exclude the transformation between the BPMN choreography and the BPMN graph.

5.1.1. BPMN graph

As said previously, we propose to abstract first the BPMN choreography in a BPMN graph. In this part we provide a formal definition of this graph.

The goal of this part is to indicate precisely the list of BPMN artefacts that we consider. We assume that a choreography description using BPMN is based on the following elements: Activities, Gates and Transitions. We recall that on choreographies, activities involve at least two participants.

Definition 5.1.1 *A BPMN choreography graph is a tuple* $C = (W, G, E, T)$ *where,* W *is the set of activities,* G *the set of gates.* E *serves to define relation between activities* $E = \{(u, v)$ *s.t.* $u, v \in$

$\{W \cup G\}\}$ *and* T *gives for each gate its type (we have for example the fork gate).* $T = \{(g,t)$ ***s.t.*** $g \in G, t \in \{1 \dots 4\}\}$

Definition 5.1.2 *A BPMN choreography* $C = (W, G, E, T)$ *is valid if given any gate* g*:*

1) *There exist two elements* e, f *of* $W \cup G$ *such that the subsets* $\{(g,e),(f,e)\} \subseteq E$ *(each gate leads to at least two paths)*

2) *In any path on which there is a gate, we have at least one activity on the left of the gate an one on the right (the gates are routers)*

We give in the definition below a short idea on "valid choreography". As we will see after, the a more general idea can be adopted.

Definition 5.1.3 *Given a BPMN choreography* $C = (W, G, E, T)$*, we define a sub-choreography initiated by node* $s \in W' \cup G'$ *as a tuple* $C' = (s, W', G', E', T')$*, where* $E' \subseteq E, G' \subseteq G, W' \subseteq W$ *and* $\forall x \in W' \cup G'$*,* $x' \in Closure(s)$ [1]

We suppose that given a Sub-choreography C', the function $(st(C))$ gives its starting node. On some sub-choreography, there exist a unique ending node (a node that can be accessed using any nodes of the combinations). We will denote by $ed(C')$ a such node.

5.1.2. QoS Evaluation Settings

Admitting the restricted BPMN above, we give here an overview for interpreting the execution of a choreography. The following points are important:

- We suppose that the execution of each activity comprises communications and computations. Computations for each activities are done on a distinguished Web server(grids, clusters, single machine).
- We suppose that the communication time is minor in the execution (we neglect it)
- We suppose that the choreography is executed in the extremal case. This means that each participant reacts directly when they have something to do.
- We do not consider interference from execution of concurrent choreographies.

The execution of a choreography then proceeds as follows: a user sends a request from an input point. This directly instantiates an activity of a choreography and leads to a cascade of executions until the treatment of the request (we arrive at a terminal state). It is important to remark that with the definitions above, a request execution is typically a path in the graph defined (a subset of E). In the next section, we will present the different possible scenarios for QoS evaluation.

The prediction of QoS mainly depends on the execution setting that is considered. We base the different settings from the flow of request execution. Thus, we identify this list of settings:

1) In the first setting, there is a choreography with one input point for initial request submission and one user. The question is to determine the mean time expected for the initial user request. In this setting the execution of a request is described by a path followed in the choreography. The interest of this setting is to provide the minimal mean expected time (an ideal setting).

[1] $Closure$ is the transitive closure function

2) In the second setting, we assume that there are many users involved in the choreography. Assuming the request arrival law is known, we want to compute both the mean execution time for solving the request and the maximal delay time that a request spend on a server. For a user submitting requests, the mean response time is significant for knowing what is the reasonable in term of delay. In a cloud computing perspective, this information is also important for designers in order to define their service level agreements policy [CGT09]. In this scenario, it is also possible to look for bottlenecks that are paths of execution where a huge amount of time is spent.

5.1.3. Reduction based approach for QoS prediction

We consider the prediction of QoS in the single request case. Here, we have a choreography C and the objective is to determine the mean time for executing a single request. We assume that the choreography designer specification has been translated in a BPMN graph and that to each activity, we have estimated execution time. Here, we will show how we can derive the mean execution time expected from the choreography execution.

For computing this time, we employ in this case the idea of graph reduction [SMO00, CSM02] adapted to BPMN. The QoS prediction is made in reducing a BPMN choreography onto basic WS flows (we proceed as in language recognition). A flow is composed by the following BPMN concepts: Activities, Transitions, Sub-choreography. To each flow we associate an interpretation that explain how QoS evolves. The reduction idea has been investigated in many works. On some cases, it was for studying the "logic" of workflow processes like compliance checking [ADW08], well formed BPMN flows and translation to BPEL [ODtHvdA07, SMO00, VVK08]. It has also been used for time prediction on workflow. In particular, it is applied in [DnP+10] to predict QoS of composition of orchestrations and in [CSM02] for predicting QoS of general workflow. We differ of these works by our specificity to BPMN, our flows and mechanism of interpretation.

For making graph reduction, we adopt the following notations:

1) We suppose that each activity has a mean execution time denoted t_a (this data has been given by the designer).

2) Given a flow B we denote its mean execution time as $t_f(B)$. When we are referring to the complete flow that we are interpreting we denote this time t_f

In the table 5.1, we present a subset of flows and their interpretations. Sub-choreographies are referred by capital letters and elements of $W \cup G$ by minimal ones. g_1 refers to a Fork gate, g_2 the exclusive choice ones and g_3 the inclusive choice ones and g_4 the merging gates. We describe flows in giving a set of interconnections between activities and sub-choreography. For example the sequence flow is given by the relations (A, B) that means we have an ending element $ed(A)$ such that $(ed(A), st(B)) \in E$.

In the notation above, we have (B, x) on a flow if and only if we have a common ending element $ed(B)$ in B. For exclusive choices, we take in input the mean probability to follow one path p. The probability for taking the other one is then $1 - p$ (because of choice exclusion). In the inclusive gate, we assume that at least one path will be taken. We also have the probability for taking exclusively one path ore the both. For the sequence looping, we admit that we can loop at list n times and at each time with a mean probability p_i. The probabilities values and the value of n is also provided in input by the choreography designer. Finally, we have two *closed* flows that are Fork/Join and Choice/Join. We introduced them to prevent conflicting interpretations.

Evaluation based on flows. With the flow, choreographies can be evaluated using the reduction strategy [CSM02]. It consists to reduce a choreography described as a graph, in subset of generic known flows for which we know the interpretation. In this reduction some conflicts might occurs. For example, in which case do we suppose that we have a fork flow instead of a sequence flow? To resolve this problem we added in table 5.1 a column ($\leq$) for indicating priorities in flows reduction. For fixing priorities, we suppose that given two flows F_1, F_2, if one can reduce F_2 to F_1, then F_1 has a higher

Flows	Description	Interpretation	$\leq$
Sequence	$Sq = \{A, B)\}$	$t_f = t_f(A) + t_f(B)$	0
Fork	$Fk = \{(a, g_1), (g_1, B), (g_1, E)\ \}$	$t_f = t_a + \max\{t_f(B), t_f(E)\}$	1
Join	$Fj = F \cup \{(B, g_1), (E, g_1), (g_1, a)\}$	$t_f = t_a + \max\{t_f(B), t_f(E)\}$	1
Exclusive Choice	$EC = \{(a, g_2), (g_2, B), (g_2, E)\}$	$t_f = t_a + pt_f(B) + (1-p)t_f(E)$	1
Inclusive Choice	$IC = \{(a, g_3), (g_3, B), (g_3, E)\}$	$t_f = t_a + p_b.t_f(B) + p_e.t_f(E) + p_{b,e} \max\{t_f(B), t_f(E)\}$	1
Sequence Looping	$Mg = \{(a, g_2), (g_2, B), (g_2, a)\}$	$t_f = (t_a + t(g_2)) \sum_{i=1}^{n} p_i.i + t_f(B)$	2
Fork/Join	$Fj = F \cup \{(B, g_1'), (E, g_1')\}$	$t_f = t_a + \max\{t_f(B), t_f(E)\}$	2
Exclusive Choice / Join	$ECj = EC \cup \{(B, g_2'), (E, g_2')\}$	$t_f = t_a + pt_f(B) + (1 - p)t_f(E) + t(g_2)$	2
Inclusive Choice / Join	$ICj = IC \cup \{(B, g_3'), (E, g_3')\}$	$t_f = t_a + p_b.t_f(B) + p_e.t_f(E) + p_{b,e} \max\{t_f(B), t_f(E)\}$	2

Table 5.1: Subset of flows

priority. Thus, the sequence flow has the lowest priority because all flows can be viewed as sequence. By the same way, the priority of Fork/Join is higher than those of the Fork and Join separately.

5.1.4. Future investigations

There are many other steps required to make effective the model described in this part. We envision to implement the algorithm above for having an automatic mechanism of prediction. To this challenge we associate the inclusion of other BPMN constructs like for example Events. Algorithmic implementation is certainly reasonable since some automatic system of reduction (in general workflow cases) exists [CSM02]. We also envision to extend the approach on multiple request cases. The intent is to propose a reduction based solution inspired of the single request submission case for other settings listed in Section 5.1.2. This implies for us to decide of a model for request submission. The Queue theory gives many alternatives in this case. For example, we can suppose that we have a function $A(t)$ that gives at each time t the number of submitted requests. We could also assume that we have a Poisson distribution for requests arrival with a rate λ. On multiple request submission, we will have probably to consider that the system contains queues. Thus, we will have to identify bottleneck (part of the execution where request remains for a long time in queue) and predict the mean expected time. Finally, the initial proposal for QoS evaluation does not take into account the fact that an activity execution might not succeed. The possibility of execution failures has to be introduced in our model. This can be done in associating to each activity a failure probability $f_i(0 \leq f_i \leq 1)$. The introduction of these probabilities must conduct us to review our description of queue evolution. For taking into account excution faults, we can also refer to studies done on stochastic QoS prediction with Workflow nets [XWHY06] or Stochastic automata networks [BFV11].

It is important to point out that the chosen approach does not involve the several possible tools and models proposed on the work package outline, however, we think that this new approach is promising.

5.2. Choreography stability & interdependency analysis

Software systems have to evolve to continuously satisfy users' needs [MD08]. However, changes to the code may result in undesirable side or ripple effects [YCM78, AB96, KM06]. A side effect is a condition that leads the software system to a state that is erroneous or violates the original assumptions/semantics as a result of a change. A ripple effect (also known as domino-effect) is a phenomenon that affects

other parts of a system on account of a change.

To cope with the previous issues, several change impact analysis techniques have been widely explored in the context of Object-Oriented systems. In particular, a considerable body of knowledge in the area relies on analyzing existing dependencies between software artefacts [AB96, ZYXX02, Mar03, SJSJ05, PFK10, KGPC10]. Another important topic in the area concerns the assessment of architectural stability, which refers to the extent to which software systems can endure changes in requirements while leaving the architecture intact [Jaz02, BE09]. Architectural stability assessment is a feature available in a variety of structural/program analysis tools, such as IBM Structural Analysis for Java (SA4J)[2] and Headway Software Structure 101[3]. Different notions of stability have also been proposed by other authors [Mar03].

As any software system, FI choreographies will also need to evolve to remain useful. Moreover, given the ULS of FI choreographies, automated analysis mechanisms should be provided in order to cope with service coupling and choreography architecture degradation. In this context, the CHOReOS dynamic development process extends Object-Oriented change-impact analysis techniques and adapts them according to the Service-Oriented Computing paradigm. The main aspects of this change to SOC paradigm are related to (i) the identification of dependencies among participants (choreography model) and among concrete services (synthesized choreography), and (ii) the analysis of the corresponding dependency graphs. Concerning (i), we apply model-to-model transformations so that we obtain a dependency graph from which we can identify dependencies. In relation to (ii), we apply stability and graph centrality measures to assess change impact.

5.2.1. Change impact analysis metrics

Within graph theory and social network analysis (SNA) [Fre79, WF94, AHS09], there are various measures of vertex centrality that determine the relative importance of such vertex within the graph (for example, how important a person is within a social network). In particular, four specific measures of centrality that have been extensively employed in SNA: (i) degree centrality, (ii) eigenvector centrality, (iv) betweenness centrality, and (v) closeness centrality (Figure 5.3). In addition, pagerank (iii) and overall stability (vi) measures have been used to assess the 'popularity' of vertexes and the density of graphs respectively.

In this section, we describe the aforementioned measures, which form the basis of the change impact analysis approach that will be employed as part of the CHOReOS dynamic process. Hence, given a graph $G := (V, E)$ with n vertices, the measures are calculated as follows.

Degree centrality. Degree centrality is defined as the number of ties that a node has. In directed graphs, there are two separate measures of degree centrality, namely indegree centrality and outdegree centrality [WF94]. Nodes with high degree centrality have higher probability of receiving and/or transmitting whatever information flows in the network, i.e. they have influence over the nodes in their neighbourhood [AHS09]. Degree centrality is a *local measure*, as only the connections of a node with its neighbour are taken into account to evaluate node's importance.

$$C_{D^-}(v) = \frac{deg^-(v)}{n-1}$$

$$C_{D^+}(v) = \frac{deg^+(v)}{n-1}$$

Eigenvector centrality. The Eigenvector centrality of a node is proportional to the sum of the centralities of the node's neighbours, so that a node can acquire high centrality either by being connected to a lot of others (as with simple degree centrality) or by being connected to others that themselves

[2] http://www.alphaworks.ibm.com/tech/sa4j

[3] http://www.headwaysoftware.com/products/?code=Structure101

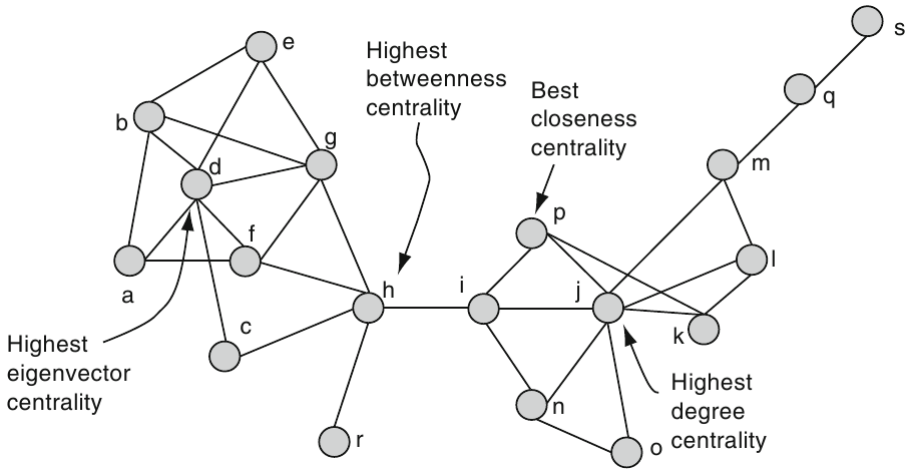


Figure 5.3: Graph centrality measures [AHS09].

are highly central [MLHH08]. The eigenvector centrality can be understood as a refined version of degree centrality, in the sense that it recursively takes into account how neighbour nodes are connected. Eigenvector centrality only exists for *undirected graphs*. This centrality measure can be computed by an iterative degree calculation procedure known as the *accelerated power method* [Hot36]. The algorithm can be described as follows.

Algorithm 3: Accelerated power method for eigenvector centrality calculation

Input : An adjacency matrix $A_{i,j}$, where $A_{i,j} = 1$ if the i^{th} node is adjacent to the j^{th} node, and $Ai, j = 0$ otherwise

Output: Eigenvector centrality value for all graph nodes

Set $C_E(v_i) = 1$ for all i ; 1

Compute $C^*_E(v_i) = \sum_j A_{i,j} * C_E(v_j)$; 2

Set λ equal to the square root of the sum of squares of each $C^*_E(v_i)$; 3

Set $C_E(v_i) = C^*_E(v_i)/\lambda$ for all i ; 4

Repeat lines 2 to 4 until λ stops changing ; 5

PageRank. PageRank algorithm is a variant of the Eigenvector centrality [AHS09]. Google's internet search engine uses PageRank to rank web pages in the result set of search queries[4]. Unlike the Eigenvector centrality, PageRank is a link analysis algorithm meant for *directed graphs*. In a simplified version of PageRank[5], the score is evenly divided among all vertices in the collection at the beginning of the computational process, which means

$$PR_0(u) = |V|^{-1}$$

where V is the set of all vertices of the graph. The PageRank computations require several iterations through the vertices set to adjust approximate PageRank values to more closely reflect the theoretical true value. At each iteration the page rank is calculated as follow.

$$PR_i(u) = \sum_{v \in Bu} \frac{PR_{i-1}(v)}{L(v)},$$

[4] PageRank is a trademark of Google.

[5] The presented formula doesn't consider the *dumping factor*; a more detailed explanation about the PageRank can be found at http://www.linksandlaw.com/technicalbackground-pagerank.htm

i.e. the PageRank value for a page u is dependent on the PageRank values for each page v in Bu (this set contains all pages linking to page u), divided by the number $L(v)$ of outbound links from page v.

Betweenness centrality. Nodes that occur on many shortest paths (a.k.a. geodesic distance) between other vertices have higher betweenness than those that do not. Hence, betweenness centrality evaluates the degree of control a node has over the information flowing in the network. Messages sent through the network frequently pass through these nodes, i.e. they act as "brokers".

$$C_B(v) = (\sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}})/[(n-1)*(n-2)],$$

where σ_{st} is the number of shortest paths from s to t, and $\sigma_{st}(v)$ is the number of shortest paths from s to t that pass through a vertex v.

Closeness centrality. The vertex closeness refers to the mean shortest path (geodesic distance) between a vertex v and all other vertices reachable from it. Closeness measures how close a node is located with respect to every other node in the network. Nodes with high closeness are able to reach most or all other nodes in the network through geodesic paths.

$$C_C(v) = \frac{|J_v|/(n-1)}{\sum_{t \in J_v} d_G(v,t)/|J_v|}$$

where J_v is the set of vertices reachable from v (a.k.a influence range of v) and $d_G(v,t)$ is the length of the geodesic path from v to t.

Overall stability. Overall stability refers to the average propagation level of changes performed in the nodes of a network. The higher the stability, the less changes tend to propagate. In object-oriented systems, this measure is employed to determine whether the overall coupling between elements is being controlled.

$$AvgImpact(G) = [\sum_{v \in V} \gamma(v)]/n^2$$

$$Stability(G) = 1 - AvgImpact(G),$$

where $\gamma(v)$ denotes the size of the transitive closure of v (graph vertices from which v can be reached).

For instance, a stability level of 70% means that a change affects, in average, 30% of all existing network nodes.

5.2.2. Analysis based on choreography roles

Choreographies can be seen as *social networks* in which participants share information and collaborate through message exchanges. In particular, ULS choreographies of the FI will originate large social networks. Centrality measures presented in the previous section offer a straightforward and yet powerful way to assess the prominence (importance) of participants in a choreography.

Identifying prominent participants is important, since they should be realized by dependable [BJRT07] and reliable services (or groups of services). Hence, adequately monitoring and testing these services should improve choreography robustness and evolvability. These services also constitute good candidates to go through integration tests. Prominence values should be recalculated every time a choreography model changes (e.g., due to new business rules), so that the set of important concrete services are kept updated.

Furthermore, the choreography model itself establishes a certain degree of coupling between roles. In fact, except in the case where a specific service realizes more than one choreography role, the coupling degree of the synthesized choreography will be greater than or equal to that of the choreography

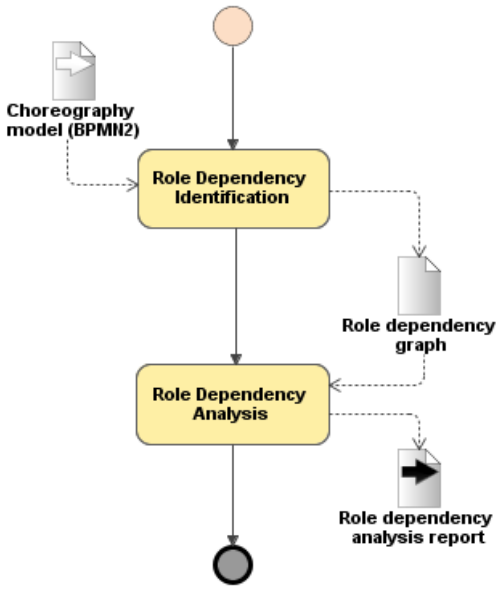


Figure 5.4: Role dependency analysis.

model. Hence, evaluating the stability of the choreography model should aid the choreography designer in designing more maintainable choreographies.

To calculate stability and centrality measures, the BPMN choreography model (Figure 2.2) must be first transformed into a directed graph denoting the dependencies between the participants (role dependencies). In this scenario, dependencies are captured through the analysis of the message exchanges defined in the BPMN choreography model [WC09]. This graph representation is very similar to the model presented in section 5.1.1, but in this case there is no need of modelling the BPMN gateways. A dependency identification technique is proposed and described in Section 5.5 of deliverable 4.1 from project CHOReOS. Figure 5.4 describes the process for role dependency analysis.

5.2.3. Analysis based on service dependencies

Dependencies among choreographed services are captured according to defined operation calls between them [WC09]. The results of the choreography synthesis process (see *peer-style specification* in Figure 2.7) provide the input for service dependency identification. The analysis based on service dependencies should be taken into account during choreography synthesis and dynamic adaptation in order to improve choreography evolvability. Figure 5.5 describes the process for service dependency analysis.

We distinguish between two kinds of service dependency analysis: service-centric analysis and choreography-wide analysis. In the next subsections, we present and discuss these two kinds of analyses.

Service-centric analysis

In the dynamic environment in which FI choreographies take place, services may malfunction or become unavailable. In this context, we apply the different centrality measures to identify services that are more likely to experience side-effects.

We first apply the closeness centrality measure. The basic idea is that specific services that are close to many others in the dependency graph are more prone to experience side-effects. Therefore, services with high closeness should be properly identified and monitored.

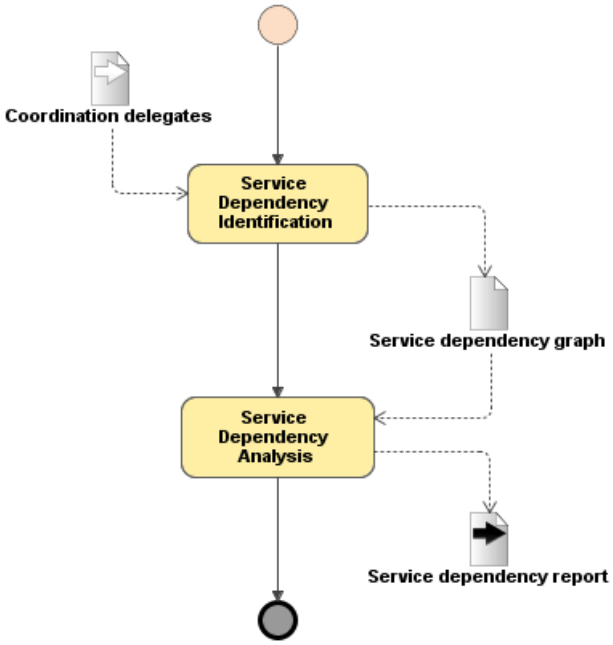


Figure 5.5: Service dependency analysis.

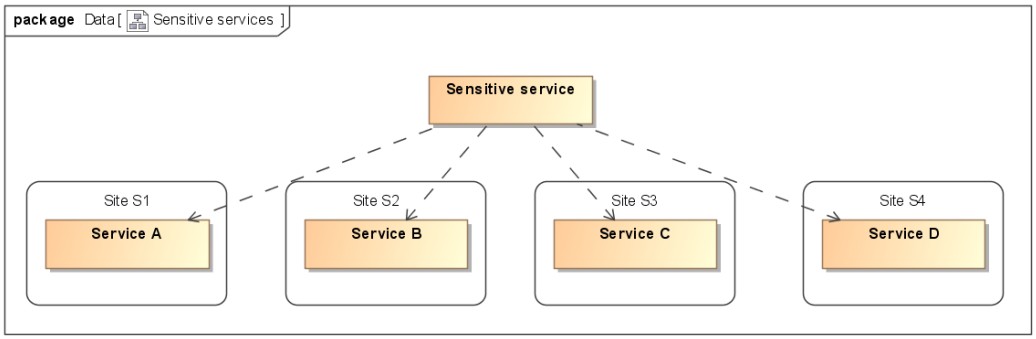


Figure 5.6: Local sensitive service in a choreography.

Furthermore, we apply the degree centrality measure. Based on such measure, we distinguish between the following kinds of choreography services:

Sensitive services. In graph theory terms, a local sensitive service represents a node that owns a high out-degree (Figure 5.6). A local sensitive service depends on many other services and is thus likely to experience side effects in the face of changes in the supplier services. Also, side effects may occur due to supplier service faults, malfunctioning or low performance. In particular, such side effects tend to increase when supplier services are deployed in different distributed sites that are accessed through different networks.

Global sensitive services, in turn, are those that have a high number of global/transitive dependencies. A sensitive service can be either local, global, or both. Global sensitive services are often affected when any other service in the choreography is changed. A high occurrence rate of global sensitive services implies that a choreography is highly unstable.

Therefore, it is crucial to have sensitive services properly identified and monitored, to maintain it working seamlessly. It is also advisable to limit the occurrence of sensitive services when synthesizing choreographies in order to improve choreography maintainability and evolvability.

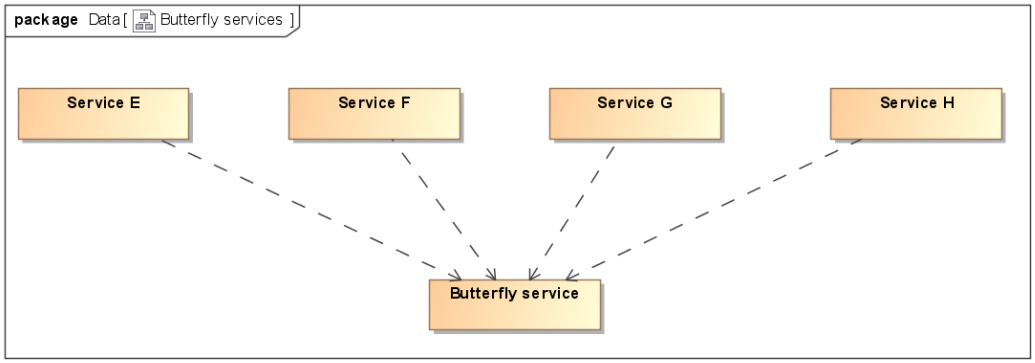


Figure 5.7: Local butterfly service in a choreography.

Butterfly services. In graph theory terms, a local butterfly service represents a node that owns a high in-degree (Figure 5.7). A local butterfly service is depended on by many other services and is thus likely to cause side effects in its clients when it goes through changes. Also, side effects may occur due to local butterfly service faults, malfunctioning or low performance.

Global butterfly services, in turn, are those that have a high number of global/transitive incoming dependencies. A butterfly service can be either local, global, or both. Global butterflies usually represent core services of the choreography and they often affect a large portion of services when changed. Changing a global butterfly must be done with caution, since it can possibly affect a large number of elements in the choreography.

Therefore, it is crucial to have butterfly services properly identified, monitored and backup up by duplicate or equivalent services. Furthermore, unit-testing this kind of service and performing integration tests involving its clients are also highly advisable.

Hub services. In graph theory terms, a local hub service represents a node that owns a high in-degree and a high out-degree at the same time (Figure 5.8). A local hub service is both a butterfly service and a sensitive service at the same time. A local hub service consists in a change propagator that amplifies the changes throughout the system, i.e., it is likely to change (since it depends on lots of services) and it is likely to spread such changes (since lots of other services depend on it).

Global hub services, in turn, are those that have a high number of both global/transitive incoming and outgoing dependencies. A hub service can be either local, global, or both. Global hubs can be very harmful to the choreography, since they often affect a large portion of the choreography when changed and are also frequently affected when any service is changed.

Therefore, hub services should be avoided when possible during choreography synthesis and adaptation.

Choreography-wide analysis

This section refers to choreography-wide kinds of interdependency analysis.

Service tangles. A tangle is a large group of nodes that are so interconnected that a change performed in any of them can transitively affect all the others. More precisely, a tangle is a strong component in the dependency graph, i.e., for each pair of vertices (u, v), there is a path from u to v. Avoiding service tangles is crucial to minimize ripple effects in the choreography.

Overall stability. The choreography overall stability measure refers to the average propagation level of changes performed in the services of a synthesized choreography. This measure aids in determining whether the overall coupling between services is being controlled, since changes do not tend to propagate in choreographies whose stability level is high. Maximizing stability is an important direction towards reducing change impact levels.

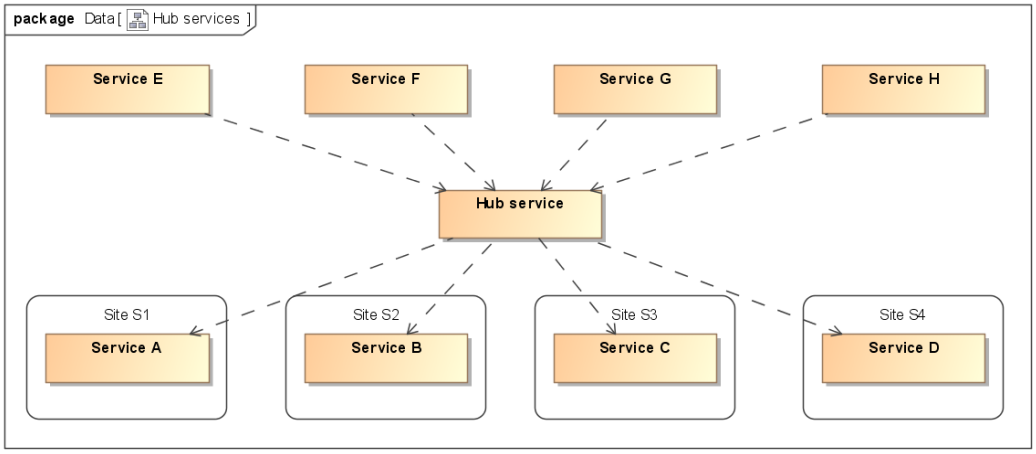


Figure 5.8: Local hub service in a choreography.

6 Conclusions and Future Work

In this deliverable, we presented an initial version of the CHOReOS dynamic development process model mainly consisting of the following (macro-)activities: (i) a domain-expert requirements specification framework for adaptable QoS-aware highly-scalable choreographies; (ii) methods and mechanisms for large scale service base management; (iii) automated choreography synthesis; (iv) choreography deployment and execution; and (v) governance V&V, monitoring and V&V configuration.

We have given an overview for each activity and specifically for activities (i) and (ii) we also provided a detailed description by considering a simple case study concerning a passenger friendly airport scenario.

According to the plan as reported in the Dow, as future work, at M24, we will mainly focus on instantiating the process model into an actual development process. To this end, we will refine each activity in the process model by exactly identifying techniques, methods and tools (to be integrated by the IDRE) for choreography design, analysis, synthesis, deployment and enactment, and we will precisely characterize the software artifacts exploited by each refined activity. Then, at M36, we will implement the identified methods and tools. As further future work, we will also define suitable M2M transformations in order to achieve as much complete integration as possible of the various process activities.

Bibliography

[AB96] R. Arnold and S. Bohner. *Software change impact analysis*. Wiley-IEEE Computer Society, 1996.

[ADW08] Ahmed Awad, Gero Decker, and Mathias Weske. Efficient compliance checking using bpmn-q and temporal logic. In Marlon Dumas, Manfred Reichert, and Ming-Chien Shan, editors, *Business Process Management, 6th International Conference, BPM 2008, Milan, Italy, September 2-4, 2008. Proceedings*, volume 5240 of *Lecture Notes in Computer Science*, pages 326–341. Springer, 2008.

[AHS09] Ajith Abraham, Aboul-Ella Hassanien, and Vclav Snsel. *Computational Social Network Analysis: Trends, Tools and Research Advances*. Springer Publishing Company, Incorporated, 1st edition, 2009.

[AMM08] Eyhab Al-Masri and Qusay H. Mahmoud. Investigating web services on the world wide web. In *Proc. of WWW '08*, pages 795–804, 2008.

[AS02] I.F. Alexander and R. Stevens. *Writing Better Requirements*. Addison-Wesley, 2002.

[AZVI11] Dionysis Athanasopoulos, Apostolos Zarras, Panos Vassiliadis, and Valérie Issarny. Mining service abstractions: Nier track. In *ICSE*, pages 944–947, 2011.

[BAP11] Antonia Bertolino, Guglielmo De Angelis, and Andrea Polini, editors. *Governance V&V policies and rules*. Number D4.1. The CHOReOS Consortium, Oct. 2011.

[BBF09] G. Blair, N. Bencomo, and R. B. France. Models@run.time. *Computer journal*, 42:22–27, 2009.

[BCH+08] Salima Benbernou, Luca Cavallaro, Mohand Said Hacid, Raman Kazhamiakin, Gabor Kecskemeti, Jean-Louis Poizat, Fabrizio Silvestri, Maike Uhlig, and Branimir Wetzstein. State of the art report, gap analysis of knowledge on principles, techniques and methodologies for monitoring and adaptation of sbas, July 2008.

[BE09] Rami Bahsoon and Wolfgang Emmerich. Architectural stability. In *Proceedings of the Confederated International Workshops and Posters on On the Move to Meaningful Internet Systems: ADI, CAMS, EI2N, ISDE, IWSSA, MONET, OnToContent, ODIS, ORM, OTM Academy, SWWS, SEMELS, Beyond SAWSDL, and COMBEK 2009*, OTM '09, pages 304–315, Berlin, Heidelberg, 2009. Springer-Verlag.

[BFV11] Kelly Rosa Braghetto, Joao Eduardo Ferreira, and Jean-Marc Vincent. From Business Process Model and Notation to Stochastic Automata Network. Technical Report (Reference Number: RT-MAC-2011-03), University of São Paulo, March 2011. [Available at: http://www.ime.usp.br/ kellyrb/files/; accessed 15-June-2011].

[BIPT09] Antonia Bertolino, Paola Inverardi, Patrizio Pelliccione, and Massimo Tivoli. Automatic synthesis of behavior protocols for composable web-services. In *Proc. of ESEC/FSE '09*, pages 141–150, 2009.

[BJRT07] Michael Butler, Cliff Jones, Alexander Romanovsky, and Elena Troubitsyna. *Rigorous Development of Complex Fault-Tolerant Systems (Lecture Notes in Computer Science)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

[BP05] A. Bertolino and A. Polini. The audition framework for testing web services interoperability. In *EUROMICRO-SEAA* [DBL05], pages 134–142.

[Bri92] Eric Brill. A simple rule-based part of speech tagger. In *ANLP*, pages 152–155, 1992.

[BYS10] Ricardo A. Baeza-Yates and Alejandro Salinger. Fast intersection algorithms for sorted sequences. In *Algorithms and Applications*, pages 45–61, 2010.

[CE00] Krzysztof Czarnecki and Ulrich W. Eisenecker. *Generative Programming*. Addison-Wesley, 2000.

[CGT09] Allan Clark, Stephen Gilmore, and Mirco Tribastone. *Service-Level Agreements for Service-Oriented Computing*, pages 21–36. Springer-Verlag, Berlin, Heidelberg, 2009.

[CHO11a] CHOReOS Project Team. Choreos middleware specification. `www.choreos.eu`, October 2011.

[CHO11b] CHOReOS Project Team. Choreos perspective on the future internet and initial conceptual model. `www.choreos.eu`, April 2011.

[CHO11c] CHOReOS Project Team, Public Project Deliverable D1.2. *CHOReOS Perspective on the Future Internet and Initial Conceptual Model*, April 2011.

[CHO11d] CHOReOS Project Team, Public Project Deliverable D1.3. *Initial Architectural Style for CHOReOS Choreographies*, October 2011.

[Com98] International Standards Organisation & International Electrotechnical Commission. *International Standard ISO 9241-11 Ergonomic requirements for office work with visual display terminals (VDTs) Part 11: Guidance on Usability*. 1998.

[COR] CORDIS ICT Programme. Service and Software Architectures, Infrastractures and Engineering (SSAI). `http://cordis.europa.eu/fp7/ict/ssai/home_en.html`.

[CSM02] Jorge Cardoso, Amit P. Sheth, and John A. Miller. Workflow quality of service. In *ICEIMT*, pages 303–311, 2002.

[DBL05] *31st EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO-SEAA 2005), 30 August - 3 September 2005, Porto, Portugal*. IEEE Computer Society, 2005.

[DHM+04] X. Dong, A. Halevy, J. Madhavan, E. Nemes, and J. Zhang. Similarity Search for Web Services. In *Proceedings of VLDB*, 2004.

[DLS05] Glen Dobson, Russell Lock, and Ian Sommerville. Qosont: a qos ontology for service-centric systems. In *EUROMICRO-SEAA* [DBL05], pages 80–87.

[DnP+10] Marlon Dumas, Luciano García-Ba nuelos, Artem Polyvyanyy, Yong Yang, and Liang Zhang. Aggregate quality of service computation for composite services. In Paul P. Maglio, Mathias Weske, Jian Yang, and Marcelo Fantinato, editors, *Service-Oriented Computing - 8th International Conference, ICSOC 2010, San Francisco, CA, USA, December 7-10, 2010. Proceedings*, volume 6470 of *Lecture Notes in Computer Science*, pages 213–227, 2010.

[Ebi11] Ebiz. The insider's guide to next-generation bpm. http://www.ebizq.net/blogs/soainaction/2010/05/mobility_versus_portability_pa.php, 2011.

[Fre79] Linton C. Freeman. Centrality in social networks: Conceptual clarification. *Social Networks*, 1:215– 239, 1979.

[Gar09] Gartner. Gartner highlights five attributes of cloud computing. http://www.gartner.com/it/page.jsp?id=1035013, 2009.

[Gil] Tom Gilb. Quantifying the qualitative. http://www.result-planning.com.

[HMY06] Gang Huang, Hong Mei, and Fu-Qing Yang. Runtime recovery and manipulation of software architecture of component-based systems. *Automated Software Engg.*, 13:257–281, April 2006.

[Hot36] Harold Hotelling. Simplified calculation of principal components. *Psychometrika*, 1:27–35, 1936. 10.1007/BF02287921.

[IBM10] IBM. Cloud services may be a game-changer for business. http://www-304.ibm.com/businesscenter/cpe/html0/190102.html, 2010.

[ISO01] ISO/IEC. *ISO/IEC 9126. Software engineering – Product quality*. ISO/IEC, 2001.

[Jaz02] Mehdi Jazayeri. On architectural stability and evolution. In *Proceedings of the 7th Ada-Europe International Conference on Reliable Software Technologies*, Ada-Europe '02, pages 13–23, London, UK, UK, 2002. Springer-Verlag.

[KGPC10] Huzefa Kagdi, Malcom Gethers, Denys Poshyvanyk, and Michael L. Collard. Blending conceptual and evolutionary couplings to support change impact analysis in source code. In *Proceedings of the 2010 17th Working Conference on Reverse Engineering*, WCRE '10, pages 119–128, Washington, DC, USA, 2010. IEEE Computer Society.

[KKZ09] Hans-Peter Kriegel, Peer Kröger, and Arthur Zimek. Clustering high-dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 3(1), 2009.

[KM06] Huzefa Kagdi and Jonathan I. Maletic. Software-change prediction: Estimated+actual. In *Proceedings of the Second International IEEE Workshop on Software Evolvability*, pages 38–43, Washington, DC, USA, 2006. IEEE Computer Society.

[Kri08] Kyriakos Kritikos. *Qos-based web service description and discovery*. PhD thesis, University of Crete, 2008.

[LJL+03] Kangchan Lee, Jonghong Jeon, Wonseok Lee, Seong-Ho Jeong, and Sang-Won Park. Qos for web services: Requirements and possible approaches. *W3C Working Group Note*, 25(November):1–9, 2003.

[LKP+08] James Lockerbie, Kristine Karlsen, Michele Puccio, Vito Morreale, and Susanna Bonura. Using requirements to define services for service-centric food traceability information systems. In *Proceedings of the 2008 International Workshop on Service-Oriented Computing Consequences for Engineering Requirements*, SOCCER '08, pages 15–23, Washington, DC, USA, 2008. IEEE Computer Society.

[Llo82] Stuart P. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2):129–136, 1982.

[LW94] B. Liskov and J.M. Wing. A Behavioral Notion of Subtyping. *ACM Transactions on Programming Languages and Systems (ACM TOPLAS)*, 16(6):1811–1841, 1994.

[Mai08] Neil Maiden. Requirements engineering. www.it.uu.se/edu/course/homepage/acsd/vt08/Neil1.pdf, 2008.

[Mar03] Robert Cecil Martin. *Agile Software Development: Principles, Patterns, and Practices*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2003.

[MB07] Onaiza Maqbool and Haroon A. Babri. Hierarchical Clustering for Software Architecture Recovery. *IEEE Transactions on Software Engineering (TSE)*, 33(11):759–780, 2007.

[MBF+90] G. Miller, R. Beckwith, C. Fellbaum, D. Gross, and K. Miller. Introduction to wordnet: An on-line lexical database. *International Journal of Lexicography*, 3(4):235–244, 1990.

[MBK+09] Nebil Ben Mabrouk, Sandrine Beauche, Elena Kuznetsova, Nikolaos Georgantas, and Valérie Issarny. Qos-aware service composition in dynamic service oriented environments. In *Middleware*, pages 123–142, 2009.

[MCvdHP08] Michele Mancioppi, Manuel Carro, Willem-Jan van den Heuvel, and Mike P. Papazoglou. Sound multi-party business protocols for service networks. In *ICSOC*, pages 302–316, 2008.

[MD08] Tom Mens and Serge Demeyer. *Software Evolution*. Springer Publishing Company, Incorporated, 1 edition, 2008.

[MLHH08] Nasrullah Memon, Henrik Legind Larsen, David L. Hicks, and Nicholas Harkiolakis. Detecting hidden hierarchy in terrorist networks: Some case studies. In *Proceedings of the IEEE ISI 2008 PAISI, PACCF, and SOCO international workshops on Intelligence and Security Informatics*, PAISI, PACCF and SOCO '08, pages 477–489, Berlin, Heidelberg, 2008. Springer-Verlag.

[Mun57] J. Munkres. Algorithms for the Assignment and Transportation Problems. *Journal of the Society for Industrial and Applied Mathematics*, 5(1):32–38, 1957.

[ODtHvdA07] Chun Ouyang, Marlon Dumas, Arthur H.M. ter Hofstede, and Wil M.P. van der Aalst. Pattern-based translation of bpmn process models to bpel web services. *International Journal of Web Services Research (JWSR)*, 5(1):42–62, 2007.

[OPF] Open Process Framework OPF. Quality requirement. http://www.opfro.org/.

[PFK10] Guenter Pirklbauer, Christian Fasching, and Werner Kurschl. Improving change impact analysis with a tight integrated process and tool. In *Proceedings of the 2010 Seventh International Conference on Information Technology: New Generations*, ITNG '10, pages 956–961, Washington, DC, USA, 2010. IEEE Computer Society.

[PHL04] Lance Parsons, Ehtesham Haque, and Huan Liu. Subspace clustering for high dimensional data: a review. *SIGKDD Explorations*, 6(1):90–105, 2004.

[PMM97] Fabio Paternò, Cristiano Mancini, and Silvia Meniconi. Concurtasktrees: A diagrammatic notation for specifying task models. In *Proceedings of the IFIP TC13 Interantional Conference on Human-Computer Interaction*, INTERACT '97, pages 362–369, London, UK, UK, 1997. Chapman & Hall, Ltd.

[PS02] Fabio Paternò and Carmen Santoro. Preventing user errors by systematic analysis of deviations from the system task model. *Int. J. Hum.-Comput. Stud.*, 56:225–245, February 2002.

[Ran03] Shuping Ran. A model for web services discovery with qos. *SIGecom Exchanges*, 4(1):1–10, 2003.

[Rea90] James Reason. *Human Error*. Cambridge University Press, 1990.

[RR99] S. Robertson and J. Robertson. *Mastering the Requirements Process*. Addison-Wesley, 1999.

[RR00] James Robertson and Suzanne Robertson. Volere: Requirements specification template. 2000.

[SC04] S-CUBE. Project web site: Secse.eng.it, 2004.

[SC08] S-CUBE. Quality reference model for service-based applications. http://www.s-cube-network.eu/results/deliverables/wp-jra-1.3/Reference_Model_for_SBA.pdf/view, March 2008.

[SC10] S-CUBE. Knowledge model. http://www.s-cube-network.eu/km, 2010.

[SJSJ05] Neeraj Sangal, Ev Jordan, Vineet Sinha, and Daniel Jackson. Using dependency models to manage complex software architecture. In *Proceedings of the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, OOPSLA '05, pages 167–176, New York, NY, USA, 2005. ACM.

[SMO00] Wasim Sadiq, Maria, and E. Orlowska. Analyzing process models using graph reduction techniques. *Information Systems*, 25:117–134, 2000.

[SQC] ISO 9126 Software Quality Characteristics. http://www.sqa.net/iso9126.html.

[SS75] J H Saltzer and M D Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, 1975.

[SW01] Mark Stevenson and Yorick Wilks. The interaction of knowledge sources in word sense disambiguation. *Computational Linguistics*, 27(3):321–349, 2001.

[Tea10] CHOReOS Project Team. Choreos state of the art, baseline, and beyond - public project deliverable d1.1, December 2010.

[Tea11a] CHOReOS Project Team. Requirements for the choreos idre, May 2011.

[Tea11b] CHOReOS Project Team. Specification of the choreos idre, October 2011.

[TSK06] P. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining*. Pearson Addison Wesley, 2006.

[UAcLR01] Algirdas Avizienis Ucla, Algirdas Avizienis, Jean claude Laprie, and Brian Randell. Fundamental concepts of dependability. Technical Report 010028, University of California, 2001.

[VVK08] Jussi Vanhatalo, Hagen Völzer, and Jana Koehler. The refined process structure tree. In *Proceedings of the 6th International Conference on Business Process Management*, BPM '08, pages 100–115, Berlin, Heidelberg, 2008. Springer-Verlag.

[W3C] W3C. *Web Services Architecture*. W3C, Technical Report. Available at http://www.w3.org/TR/ws-arch/.

[W3C04] W3C. *XML Schema Part 2: Datatypes Second Edition*. W3C, Technical Report, October 2004. Available at http://www.w3.org/TR/xmlschema-2/.

[W3C10] W3C. Qos for web services: Requirements and possible approaches. http://www.w3c.or.kr/kr-office/TR/2003/ws-qos/, 2010.

[WC09] Shuying Wang and Miriam A. M. Capretz. A dependency impact analysis model for web services evolution. In *Proceedings of the 2009 IEEE International Conference on Web Services*, ICWS '09, pages 359–365, Washington, DC, USA, 2009. IEEE Computer Society.

[WF94] S. Wasserman and K. Faust. *Social Network Analysis: Methods and Applications (Structural Analysis in the Social Sciences)*. Cambridge University Press, 1994.

[Wik10] Wikipedia. Serviceability (computer). http://en.wikipedia.org/wiki/Serviceability_(computer), 2010.

[WS03] Yiqiao Wang and Eleni Stroulia. Semantic structure matching for assessing web-service similarity. In *ICSOC*, pages 194–207, 2003.

[WSB93] B. J. Wielinga, A. Th. Schreiber, and J. A. Breuker. Readings in knowledge acquisition and learning. chapter KADS: a modelling approach to knowledge engineering, pages 92–116. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[XWHY06] Yunni Xia, H. P. Wang, Y. Huang, and L. Yuan. A stochastic model for workflow qos evaluation. *Sci. Program.*, 14:251–265, December 2006.

[YCM78] SS Yau, JS Collofello, and T. MacGregor. Ripple effect analysis of software maintenance. In *The IEEE Computer Society's Second International Computer Software and Applications Conference, 1978. COMPSAC'78*, pages 60–65, 1978.

[YM94] Eric S. K. Yu and John Mylopoulos. Understanding "why" in software process modelling, analysis, and design. In *ICSE*, pages 159–168, 1994.

[ZAM] K. Zachos, A.Kounkou, and N.A.M. Maiden. Exploiting codified user task knowledge to discover services. Technical report, School of informatics, City University London.

[ZLHM11] Konstantinos Zachos, James Lockerbie, Brian Hughes, and Peter Matthews. Towards a framework for describing cloud service characteristics for use by chief information officers. In *Proceedings of the 2011 International Workshop on Service-Oriented Computing Consequences for Engineering Requirements (to appear)*, SOCCER '11, Washington, DC, USA, 2011. IEEE Computer Society.

[ZYXX02] Jianjun Zhao, Hongji Yang, Liming Xiang, and Baowen Xu. Change impact analysis to support architectural evolution. *Journal of Software Maintenance*, 14:317–333, September 2002.

Appendix A: Measures and Metrics Defined

TEMPLATE FOR SERVICE MEASUREMENT

Characteristic			
Description			
Measure	**Name**		
	Comments		
	Description		
	Metric	**XSD Type**	
		Unit	
		Value Range	
		Source	
		Calculation Algorithm	
		Coding	
	Indicator	**Question**	
		SR	

Characteristic	Access Control
Description	To ensure that access by users and client applications is controlled: users and client applications are identified; identities are properly verified; users and client applications can only access data and services for which they have been properly authorized. [OPF]

Characteristic	Accuracy	
Description	Definition of the error rate produced by the service calculated on the basis of the expected results. [SC10, SC08] Web services should be provided with high accuracy. The number of errors that the service generates over a time interval should be minimized. [W3C10]	
Measure	**Name**	NumberOfDecimalPlaces [DLS05]

	Comments	Operation-specific	
	Description	For computed numerical results, to what number of decimal places is the result accurate?	
	Metric	**XSD Type**	int
		Unit	Unitless
		Value Range	0 – ∞
		Source	
		Calculation Algorithm	
		Coding	

Characteristic	Adaptability		
Description	Characterizes the ability of the system to change to new specifications or operating environments [SQC]		
Measure	**Name**	MeanTimeToAdapt [Gil]	
	Description	Time needed to adapt from a defined [Initial State] to another defined [Final State] using defined [Means].	
	Metric	**XSD Type**	float
		Unit	Units of time
		Value Range	0.0 – ∞
		Source	
		Calculation Algorithm	
		Coding	
Measure	**Name**	PersonMonthsToAdapt [Gil]	
	Description	The amount of people needed to bring about specific changes	
	Metric	**XSD Type**	float
		Unit	Unitless
		Value Range	0.0 – ∞

		Source	
		Calculation Algorithm	
		Coding	
Measure	**Name**	CostsToAdapt [Gil]	
	Comments	Currency-specific	
	Description	The costs needed to bring about specific changes	
	Metric	**XSD Type**	float
		Unit	Unitless
		Value Range	0.0 – ∞
		Source	
		Calculation Algorithm	
		Coding	

Characteristic	Availability		
Description	The web service should be ready (i.e., available) for immediate consumption. This availability is the probability that the system is up and related to reliability. Time-to-Repair (TTR) is associated with availability. TTR represents the time it takes to repair the web service. The service should be available immediately when it is invoked [W3C10].		
Measure	**Name**	AvailabilityAsPercentageUptime [DLS05]	
	Description	What percentage of some period T (often a year) of the system's lifetime has been uptime.	
	Metric	**XSD Type**	Float
		Unit	%
		Value Range	0.0 – 100.0 inclusive
		Source	
		Calculation Algorithm	
		Coding	

Measure	**Name**	ProbabilityOfAvailabilityOnDemand [DLS05]	
	Description	The probability that the system is up at a random instant of time.	
	Metric	**XSD Type**	Float
		Unit	Unitless
		Value Range	0.0 – 1.0 inclusive
		Source	
		Calculation Algorithm	
		Coding	

Characteristic	Awareness/Visibility
Description	The level of awareness and predictability of upcoming changes. This may include a need for the service provider to inform of changes to the provision of a service and/or its functionality

Characteristic	Capacity
Description	The limit of the number of simultaneous requests which should be provided with guaranteed performance. Web services should support the required number of simultaneous connections. [W3C10]

Characteristic	Dependability
Description	Dependability of a computing system is the ability to deliver service that can justifiably be trusted. [SC10, UAcLR01]
Comments	In practice dependability is made up of the characteristics reliability, availability, resilience and recoverability, with other characteristics also having auxiliary effect. There are no direct metrics of dependability.

Characteristic	Elasticity
Description	The ability of a service to scale capacity up or down as the consumer demands at the speed of full automation (which may be seconds for some services and hours for others). Elasticity is associated with not only scale but also an economic model that enables scaling in both directions

	in an automated fashion. This means that services scale on demand to add or remove resources as needed. [Gar09]

Characteristic	Flexibility
Description	To respond to changing resource requirements, business demands, IT administration needs on routine tasks etc. [IBM10] Refers to the capability of the service to behave in an acceptable way in anomalous or unexpected situations or when the context changes. [SC10, SC08]

Characteristic	Integrity		
Description	Integrity for web services should be provided so that a system or component can prevent unauthorized access to, or modification of, computer programs or data. There can be two types of integrity: data integrity and transactional integrity. Data integrity defines whether the transferred data is modified in transit. Transactional integrity refers to a procedure or set of procedures, which is guaranteed to preserve database integrity in a transaction. [W3C10]		
Measure	**Name**	ThreatPrevention [Gil]	
	Comments		
	Description	Probability to detect/prevent/capture a defined [Attack] under defined [Conditions].	
	Metric	**XSD Type**	Float
		Unit	%
		Value Range	0.0 – 100.0 inclusive
		Source	
		Calculation Algorithm	
		Coding	

Characteristic	Interoperability
Description	The ability of a software component to interact with other components or systems. [SQC] To ensure the service interoperates with other specified applications and components (overall composite business

	service) - to pass necessary data, receive data, use data it receives, and minimise integration defects [OPF]		
Measure	**Name**	StandardsComplience [RR00]	
	Description	The percentage the service complies fully with interoperability standards	
	Metric	**XSD Type**	Float
		Unit	%
		Value Range	0.0 – 100.0 inclusive
		Source	
		Calculation Algorithm	
		Coding	

Characteristic	Maintainability		
Description	The ability to identify and fix a fault within a software component. In some software quality models this characteristic is referenced as supportability. Maintainability is impacted by code readability or complexity as well as modularization. Anything that helps with identifying the cause of a fault and then fixing the fault is the concern of maintainability. [SQC]		
Measure	**Name**	MeanTimeToRepair [DLS05]	
	Comments		
	Description	The arithmetic mean of the time taken to recover from system failures.	
	Metric	**XSD Type**	Float
		Unit	Units of time
		Value Range	0.0 - ∞
		Source	
		Calculation Algorithm	
		Coding	
Measure	**Name**	MeanTimeToRecover [DLS05]	

	Comments		
	Description	The arithmetic mean of the time taken to recover from all forms of system downtime (e.g. upgrades, applying patches and other admin as well as repairs)	
	Metric	**XSD Type**	Float
		Unit	Units of time
		Value Range	0.0 - ∞
		Source	
		Calculation Algorithm	
		Coding	
Measure	**Name**	MaintainabilityIndex [DLS05]	
	Comments		
	Description	A figure determined from a polynomial equation involving a set of predictor variables. The predictor variables are each a combination of a weighted coefficient and an independent metric (e.g. lines of code, cyclomatic complexity, etc.)	
	Metric	**XSD Type**	Float
		Unit	Unitless
		Value Range	0.0 - ∞
		Source	
		Calculation Algorithm	
		Coding	

Characteristic	Mobility
Description	While portability implies the ability to migrate from one environment to another without modification, mobility allows for cross-platform deployment. Mobility should require no recompilation, no retargeting of the application itself while portability may, in fact, require both [SC10].

Characteristic	Operability		
Description	The degree to which the service enables its operators to perform their tasks [OPF] in a given environment. [SQC]		
Measure	**Name**	MeanTimeToComplete [DLS05]	
	Comments		
	Description	The arithmetic mean of the time taken to complete an operation over a specified time.	
	Metric	**XSD Type**	Float
		Unit	Units of time
		Value Range	0.0 - ∞
		Source	
		Calculation Algorithm	
		Coding	
Measure	**Name**	ChangeSuccessRatio [ISO01]	
	Comments		
	Description	Can user operate software service without failures after maintenance?	
	Metric	**XSD Type**	Float
		Unit	Unitless
		Value Range	0.0 - ∞
		Source	
		Calculation Algorithm	Observe behaviour of user or maintainer who is operating software system after maintenance. Count failures which user or maintainer encountered during operating software before and after maintenance.
		Coding	X= Na / Ta Y = { (Na / Ta) / (Nb / Tb) } Na = Number of cases which user encounters failures during operation after software was changed

			Nb = Number of cases which user encounters failures during operation before software is changed Ta = Operation time during specified observation period after software is changed Tb = Operation time during specified observation period before software is changed

Characteristic	Performance		
Description	The performance of a web service represents how fast a service request can be completed. It can be measured in terms of throughput, response time, latency, execution time, and transaction time, and so on. In general, high quality web services should provide higher throughput, faster response time, lower latency, lower execution time, and faster transaction time. [W3C10]		
Measure	**Name**	MeanInvocationRate [DLS05]	
	Comments		
	Description	The arithmetic mean of the number of invocations in a particular time frame.	
	Metric	**XSD Type**	Float
		Unit	Units of frequency
		Value Range	0.0 – ∞
		Source	
		Calculation Algorithm	
		Coding	
Measure	**Name**	MeanCPUUtilization [DLS05]	
	Comments		
	Description	The arithmetic mean of CPU utilization over time and potentially over a number of systems in a cluster.	
	Metric	**XSD Type**	Float
		Unit	Percent
		Value Range	0.0 – 100.0 inclusive

		Source	
		Calculation Algorithm	
		Coding	
Measure	**Name**	UpstreamLatencyStandardDeviation [DLS05]	
	Comments		
	Description	The standard deviation of experienced upstream network latency over some period. This metric assumes that latency varies randomly.	
	Metric	**XSD Type**	Float
		Unit	Units of time
		Value Range	0.0 – ∞
		Source	
		Calculation Algorithm	
		Coding	

Characteristic	Portability		
Description	The ease and speed at which a service can be ported to specific new environments when necessary, including the minimisation of porting costs and schedules [OPF]		
Measure	**Name**	StandardsComplience [RR00]	
	Comments		
	Description	The percentage the service complies fully with portability standards	
	Metric	**XSD Type**	Float
		Unit	%
		Value Range	0.0 – 100.0 inclusive
		Source	
		Calculation Algorithm	Count the number of items requiring compliance that have been met and compare with the number of items requiring

			compliance in the specification.
		Coding	X = 1- A / B A= Number of portability compliance items specified that have not been implemented during testing B= Total number of portability compliance items specified
Measure	**Name**	CostsToPort [Gil]	
	Comments	Currency-specific	
	Description	The cost to transport from a defined [Initial Environment] to a defined [Target Environment] using defined [Means].	
	Metric	**XSD Type**	Float
		Unit	Unitless
		Value Range	0.0 – ∞
		Source	
		Calculation Algorithm	
		Coding	

Characteristic	Privacy		
Description	Ensure unauthorised individuals and programs do not gain access to sensitive data and communications. “Need to know basis” [OPF]		
Measure	**Name**	UnauthorizedAccessRate [Gil]	
	Comments		
	Description	The system shall be exposed to less than [maxvalue] unauthorized accesses in time [period of time] of operation under normal circumstances.	
	Metric	**XSD Type**	Float
		Unit	Unitless
		Value Range	0 - ∞
		Source	
		Calculation Algorithm	

		Coding	

Characteristic	Recoverability		
Description	The ability to bring back a failed system to full operation, including data and network connections. [SQC] Recovery is a process of restoring the application after failing to perform one or more of its functions to fully satisfactory execution by any means other than replacement of the entire application. [SC10, BCH+08]		
Measure	**Name**	MeanRecoveryTime [ISO01]	
	Comments		
	Description	What is the average time the system takes to complete recovery from initial partial recovery?	
	Metric	**XSD Type**	Float
		Unit	Units of time
		Value Range	0.0 - ∞
		Source	
		Calculation Algorithm	Measure the full recovery times for each of the time the system was brought down during the specified trial period and compute the mean time.
		Coding	X= Sum(T) / B T= Time to recovery downed software system at each opportunity N= Number of cases which observed software system entered into recovery
Measure	**Name**	RecoveryLikelihood [Mai08]	
	Comments		
	Description	The likelihood of recovery according to specified failure types	

	Metric	**XSD Type**	
		Unit	%
		Value Range	0.0 – 100.0 inclusive
		Source	
		Calculation Algorithm	
		Coding	

Characteristic	Reliability		
Description	The ability of a web service to perform its required functions under stated conditions for a specified time interval. The reliability is the overall measure of a web service to maintain its service quality. The overall measure of a web service is related to the number of failures per day, week, month, or year. Reliability is also related to the assured and ordered delivery for messages being transmitted and received by service requestors and service providers. [W3C10]		
Measure	**Name**	ProbabilityOfFailureOnDemand [DLS05]	
	Comments		
	Description	In a given period, for a given operational profile what is the probability that the service will fail (i.e. deviate from specification or in the absence of a specification deviate from expected behaviour)?	
	Metric	**XSD Type**	float
		Unit	unitless
		Value Range	0.0 – 1.0 inclusive
		Source	
		Calculation Algorithm	
		Coding	
Measure	**Name**	RateOfOccurrenceOfFailures [DLS05]	
	Comments		
	Description	The (mean) rate at which failure has occurred up until time t (usually the current time)	
	Metric	**XSD Type**	

		Unit	unitless
		Value Range	0.0 - ∞
		Source	
		Calculation Algorithm	
		Coding	
Measure	**Name**	FailureIntensity [DLS05]	
	Comments		
	Description	Closely related to ROCOF, this metric gives the probability of failure in an arbitrary period t1 to t1+Δt of a system's life.	
	Metric	**XSD Type**	
		Unit	unitless
		Value Range	0.0 – 1.0 inclusive
		Source	
		Calculation Algorithm	
		Coding	
Measure	**Name**	DefaultDensity [DLS05]	
	Comments		
	Description	The number of defects (non-conformaces to specification) per thousand lines of code.	
	Metric	**XSD Type**	
		Unit	unitless
		Value Range	0.0 - ∞
		Source	
		Calculation Algorithm	
		Coding	
Measure	**Name**	MeanTimeBetweenFailures [DLS05]	
	Comments		

	Description	The arithmetic mean of the time between failures	
	Metric	**XSD Type**	
		Unit	Units of time
		Value Range	0.0 - ∞
		Source	
		Calculation Algorithm	Count the number of failures occurred during a defined period of operation and compute the average interval between the failures.
		Coding	a) X = T1 / A b) Y = T2 / A T1 = operation time T2 = sum of time intervals between consecutive failure occurrences A = total number of actually detected failures (Failures occurred during observed operation time)

Characteristic	Replaceability		
Description	Denotes the possibility to substitute a participant in a business protocol without violating interoperability with a set of participants [MCvdHP08]. At the infrastructure level, replaceability means the transparent replacement of equivalent resources, e.g. on failures, optimization actions, or for load balancing. Requires monitoring and adaptation techniques to quickly identify alternatives for possible replacements [SC10]		
Measure	**Name**	ContinuedUseOfData [ISO01]	
	Comments		
	Description	Can user or maintainer easily continue to use the same data after replacing this service to previous one? Is software service migration going on successfully?	
	Metric	**XSD Type**	float

		Unit	percent
		Value Range	0.0 – 100.0 inclusive
		Source	
		Calculation Algorithm	Observe user's or maintainer's behaviour when user is replacing service to previous one
		Coding	X = A / B A = number of data which are used in other software to be replaced and are confirmed that they are able to be continuously used B = number of data which are used in other software to be replaced and planned to be continuously reusable
Measure	**Name**	UserSupportFunctionalConsistency [ISO01]	
	Comments		
	Description	How consistent are the new components with existing user interface?	
	Metric	**XSD Type**	float
		Unit	unitless
		Value Range	0.0 – ∞
		Source	
		Calculation Algorithm	Observe the behaviour of the user and ask the opinion.
		Coding	X= 1 - A1 / A2 A= Number of new functions which user found unacceptably inconsistent with the user's expectation B= Number of new functions

Characteristic	Scalability
Description	Scalability represents the capability of increasing the computing capacity of service provider's computer system and system's ability to process more users' requests, operations or transactions in a given time interval. It is also related to performance. Web services should be scalable in

	terms of the number operations or transactions supported. [W3C10]

Characteristic	Security		
Description	Security is the protection of both a computer system and its data against unauthorized access, alteration or denial of use – i.e. occurring contrary to the desire of the person who controls the information, or the constraints supposedly enforced by the system even though the intruder may be an otherwise legitimate user of the computer. [SS75] Security for services ([LJL+03,Ran03,Kri08]) means providing authentication, authorization, confidentiality, traceability/auditability, accountability, data encryption, and non-repudiation. Safety and integrity are also considerations [SC10, UAcLR01]		
Measure	**Name**	NumberOfKnownVulnerabilities [DLS05]	
	Comments	It is not clear whether a high number of vulnerabilities makes something less or more secure. Clearly, it is important to also know how many have been patched for (this is therefore also included as a metric) as well as the age of the product.	
	Description	The total number of known vulnerabilities.	
	Metric	**XSD Type**	int
		Unit	Unitless
		Value Range	0 – ∞
		Source	
		Calculation Algorithm	
		Coding	
Measure	**Name**	NumberOfPatchedVulnerabilities [DLS05]	
	Comments	Ideally this should match the number of know vulnerabilities.	
	Description	The total number of known vulnerabilities.	
	Metric	**XSD Type**	int
		Unit	Unitless
		Value Range	0 – ∞
		Source	

		Calculation Algorithm	
		Coding	
Measure	**Name**	NumberOfCompromises [DLS05]	
	Comments		
	Description	The total number of known security compromises in a given period	
	Metric	**XSD Type**	int
		Unit	Unitless
		Value Range	0 – ∞
		Source	
		Calculation Algorithm	
		Coding	

Characteristic	Service Response Time		
Description	The time that passes while the service is completing one complete transaction. [SC10, SC08] Ensure that service does not make a user wonder whether or not it has received the user's request. Ensure that the service deals with time-critical inputs before their information is obsolete. [OPF]		
Measure	**Name**	MeanRoundTripTime [DLS05]	
	Comments		
	Description	Ignoring computation time, what is the arithmetic mean of the time it takes for a message to travel from client to service and back.	
	Metric	**XSD Type**	float
		Unit	Units of time
		Value Range	0.0 - ∞
		Source	
		Calculation Algorithm	
		Coding	

Measure	**Name**	MeanUpstreamDelay [DLS05]	
	Comments		
	Description	Ignoring computation time, what is the arithmetic mean of the time it takes for a message to travel from client to service.	
	Metric	**XSD Type**	float
		Unit	Units of time
		Value Range	0.0 - ∞
		Source	
		Calculation Algorithm	
		Coding	
Measure	**Name**	MeanDownstreamDelay [DLS05]	
	Comments		
	Description	Ignoring computation time, what is the arithmetic mean of the time it takes for a message to travel from service to client.	
	Metric	**XSD Type**	float
		Unit	unitless
		Value Range	0.0 - ∞
		Source	
		Calculation Algorithm	
		Coding	

Characteristic	Serviceability
Description	The ability of technical support personnel to install, configure, and monitor computer products, identify exceptions or faults, debug or isolate faults to root cause analysis, and provide hardware or software maintenance in pursuit of solving a problem and restoring the product into service [Wik10]

Characteristic	Suitability		
Description	The degree to which users find that the something to be suitable for the performance of their tasks [OPF]		
Measure	**Name**	FunctionalAdequacy [ISO01]	
	Description	How adequate are the evaluated functions?	
	Metric	**XSD Type**	Float
		Unit	percent
		Value Range	0.0 – 100.0 inclusive
		Source	
		Calculation Algorithm	Number of functions that are suitable for performing the specified tasks comparing to the number of function evaluated.
		Coding	X=1-A/B A= Number of functions in which problems are detected in evaluation B= Number of functions evaluated

Characteristic	Testability		
Description	Characterizes the effort needed to verify (test) a system change [SQC]. The degree to which something facilitates the creation and execution of successful tests. Ensure that a service is feasible to test and can be tested effectively and efficiently, to minimise the cost of testing and maximize the defects that can be found by testing. [OPF]		
Measure	**Name**	AvailabilityOfBuiltInTestFunction [ISO01]	
	Description	Can user and maintainer easily perform operational testing without additional test facility preparation?	
	Metric	**XSD Type**	Float
		Unit	percent
		Value Range	0.0 – 100.0 inclusive

		Source	
		Calculation Algorithm	Observe behaviour of user or maintainer who is testing software system after maintenance.
		Coding	X= A / B A= Number of cases in which maintainer can use suitably built-in test function B= Number of cases of test opportunities

Characteristic	Throughput		
Description	The number of web service requests served in a given time interval. [W3C10] Throughput can be further distinguished into input-data-throughput (arrival rate of user data in the input channel), communication throughput (user data output to a channel) and processing throughput (amount of data processed). [SC10, SC08]		
Measure	**Name**	MaximumNetworkBandwidth [DLS05]	
	Comments	In practice this would not form part of a specification of a service as it is specific to a particular netwok route. It may form part of a requirement or agreement at some other level however.	
	Description	What is the network bandwidth available between client and service?	
	Metric	**XSD Type**	Float
		Unit	Unitless
		Value Range	0.0 – ∞ inclusive
		Source	
		Calculation Algorithm	
		Coding	
Measure	**Name**	SOAPThroughput [DLS05]	
	Comments	SOAP Protocol in Clouds?	
	Description	An indication of the number of SOAP transactions executable per some specific time frame.	
	Metric	**XSD Type**	Float

		Unit	Units of frequency (perSecond, perMinute, etc.)
		Value Range	0.0 – ∞ inclusive
		Source	
		Calculation Algorithm	
		Coding	
Measure	**Name**	MaximumInvocationRate [DLS05]	
	Comments	Operation-specific	
	Description	An inidication for a specific operation how many call are supported in some specific time frame.	
	Metric	**XSD Type**	Float
		Unit	Units of frequency (perSecond, perMinute, etc.)
		Value Range	0.0 – ∞ inclusive
		Source	
		Calculation Algorithm	
		Coding	

Characteristic	Usability		
Description	Ease with which a specified set of users are able to effectively use the service [OPF]. Usability collects all those quality attributes that can be measured subjectively according to user feedback. [SC10, Com98]		
Measure	**Name**	TransactionErrorTrial [Mai08]	
	Comments		
	Description	Measures the number of transactions requiring correction, from user log data, over the total number of transactions in a session	
	Metric	**XSD Type**	Int
		Unit	Unitless

		Value Range	0 – ∞
		Source	
		Calculation Algorithm	
		Coding	
Measure	**Name**	UsabilityCompliance [ISO01]	
	Comments		
	Description	How completely does the service adhere to the standards, conventions, style guides or regulations relating to usability?	
	Metric	**XSD Type**	Float
		Unit	Percent
		Value Range	0 – 100.0 inclusive
		Source	
		Calculation Algorithm	Specify required compliance items based on standards, conventions, style guides or regulations relating to usability.
		Coding	$X = 1 - A / B$ A= Number of usability compliance items specified that have not been implemented during testing B= Total number of usability compliance items specified

Characteristic	User Efficiency		
Description	Resources expended in relation to the accuracy and completeness with which users achieve their goals. [SC10, SC08]		
Measure	**Name**	RelativeUserEfficiency [ISO01]	
	Description	How efficient is a user compared to an expert?	
	Metric	**XSD Type**	Float
		Unit	Percent

		Value Range	0.0 – 100.0 inclusive
		Source	
		Calculation Algorithm	User Test
		Coding	Relative user efficiency X = A / B A = ordinary user's task efficiency B = expert user's task efficiency

Characteristic	User Errors		
Description	The degree to which something minimizes the number of errors that its users make [OPF]. A User Error is an occasion in which a human operator's planned sequence of mental or physical activities with a system fails to achieve its intended outcome, and cannot be attributed to the intervention of some chance agency. [SC10, Rea90]		
Measure	**Name**	ErrorRate [Gil]	
	Comments		
	Description	Number of Erroneous Transactions requiring correction each session.	
	Metric	**XSD Type**	Int
		Unit	Unitless
		Value Range	0 – ∞
		Source	
		Calculation Algorithm	
		Coding	
Measure	**Name**	UserErrorCorrection [Gil]	
	Comments		
	Description	How frequently does the user successfully correct input errors?	
	Metric	**XSD Type**	Float
		Unit	Percent
		Value Range	0.0 – 100.0 inclusive

		Source	
		Calculation Algorithm	Conduct user test and observe user behaviour.
		Coding	X= A / B A= Number of input errors which the user successfully corrects B= Number of attempts to correct input errors

Characteristic	User Satisfaction		
Description	The degree to which users are satisfied with something and consider it to be beneficial to them [OPF]. Freedom from discomfort and positive attitudes towards the use of the service. [SC10, SC08]		
Measure	**Name**	ChangeCycleEfficiency [ISO01]	
	Comments		
	Description	Can the user's problem be solved to his satisfaction within an acceptable time scale?	
	Metric	**XSD Type**	Float
		Unit	Unit of time
		Value Range	0 – ∞
		Source	
		Calculation Algorithm	Monitor interaction between user and supplier. Record the time taken from the initial user's request to the resolution of problem.
		Coding	Average Time : Tav = Sum(Tu) / N Tu= Trc - Tsn Tsn= Time at which user finished to send request for maintenance to supplier with problem report Trc= Time at which user received the revised version release (or status report) N= Number of revised versions